
Pós-Graduação em Ciência da Computação

# “HdSC: Modelagem de Alto Nível para Simulação Nativa de Plataformas com Suporte ao Desenvolvimento de HdS”

**Por**

***BRUNO OTÁVIO PIEDADE PRADO***

**Tese de Doutorado**


Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE
2014

*Universidade Federal de Pernambuco*

*CENTRO DE INFORMÁTICA*

*PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO*

***BRUNO OTÁVIO PIEDADE PRADO***

# *"HDSC: MODELAGEM DE ALTO NÍVEL PARA SIMULAÇÃO NATIVA DE PLATAFORMAS COM SUPORTE AO DESENVOLVIMENTO DE HDS"*

Este trabalho foi apresentado à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Doutor em Ciência da Computação.

*ORIENTADOR(A):* ***Profa. Edna Natividade da Silva Barros***

*RECIFE*
*2014*

Catalogação na fonte
Bibliotecária Jane Souto Maior, CRB4-571

P896h Prado, Bruno Otávio Piedade
HdSC: modelagem de alto nível para simulação nativa de plataformas com suporte ao desenvolvimento de HdS / Bruno Otávio Piedade Prado. – Recife: O Autor, 2014.
387 p.: il., fig., tab.

Orientador: Edna Natividade da Silva Barros.
Tese (Doutorado) – Universidade Federal de Pernambuco. CIn. Ciência da Computação, 2014.
Inclui referências e apêndices.

1. Engenharia da computação. 2. Modelagem de sistemas. 3. Sistemas embarcados. 4. Avaliação de desempenho. I. Barros, Edna Natividade da Silva (orientadora). I. Título.

621.39 CDD (22. ed.) UFPE- MEI 2014-152

Tese de Doutorado apresentada por **Bruno Otávio Piedade Prado** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título "**HdSC: Modelagem de Alto Nível para Simulação Nativa de Plataformas com Suporte ao Desenvolvimento de HdS**" orientada pela **Profa. Edna Natividade da Silva Barros** e aprovada pela Banca Examinadora formada pelos professores:

_____
Prof. Abel Guilhermino da Silva Filho
Centro de Informática / UFPE

_____
Prof. Ricardo Massa Ferreira Lima
Centro de Informática / UFPE

_____
Prof. Manoel Eusebio de Lima
Centro de Informática / UFPE

_____
Prof. Antônio Augusto Fröhlich
Departamento de Informática e Estatística / UFSC

_____
Prof. Luigi Carro
Departamento de Informática Aplicada / UFRGS

Visto e permitida a impressão.
Recife, 27 de agosto de 2014.

_____
**Profa. Edna Natividade da Silva Barros**
Coordenadora da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

*Dedico este trabalho a Deus e a minha família, sem os quais nada disto seria possível nem faria sentido.*

*"Justiça tardia nada mais é do que a injustiça institucionalizada."*

**Rui Barbosa**

# *Resumo*

Com os grandes avanços recentes dos sistemas computacionais, houve a possibilidade de ascensão de dispositivos inovadores, como os modernos telefones celulares e tablets com telas sensíveis ao toque. Para gerenciar adequadamente estas diversas interfaces é necessário utilizar o software dependente do hardware (HdS), que é responsável pelo controle e acesso a estes dispositivos. Além deste complexo arranjo de componentes, para atender a crescente demanda por mais funcionalidades integradas, o paradigma de multiprocessamento vem sendo adotado para aumentar o desempenho das plataformas.

Devido à lacuna de produtividade de sistemas, tanto a indústria como a academia têm pesquisado processos mais eficientes para construir e simular sistemas cada vez mais complexos. A premissa dos trabalhos do estado da arte está em trabalhar com modelos com alto nível de abstração e de precisão que permitam ao projetista avaliar rapidamente o sistema, sem ter que depender de lentos e complexos modelos baseados em ISS.

Neste trabalho é definido um conjunto de construtores para modelagem de plataformas baseadas em processadores, com suporte para desenvolvimento de HdS e reusabilidade dos componentes, técnicas para estimativa estática de tempo simulado em ambiente nativo de simulação e suporte para plataformas multiprocessadas. Foram realizados experimentos com aplicações de entrada e saída intensiva, computação intensiva e multiprocessada, com ganho médio de desempenho da ordem de 1.000 vezes e precisão de estimativas com erro médio inferior a 3%, em comparação com uma plataforma de referência baseada em ISS.

**Palavras-chave:** Exploração de espaço de projeto. Validação e análise de modelos. Hardware-dependent Software (HdS). Protótipo virtual. Arquiteturas multiprocessadas.

# *Abstract*

The amazing advances of computer systems technology enabled the rise of innovative devices, such as modern touch sensitive cell phones and tablets. To properly manage these various interfaces, it is required the use of the Hardware-dependent Software (HdS) that is responsible for these devices control and access. Besides this complex arrangement of components, to meet the growing demand for more integrated features, the multiprocessing paradigm has been adopted to increase the platforms performance.

Due to the system design gap, both industry and academia have been researching for more efficient processes to build and simulate systems with this increasingly complexity. The premise of the state of art works is the development of high level of abstraction and precise models to enable the designer to quickly evaluate the system, without having to rely on slow and complex models based on instruction set simulators (ISS).

This work defined a set of constructors for processor-based platforms modeling, supporting HdS development and components reusability, techniques for static simulation timing estimation in native environment and support for multiprocessor platforms. Experiments were carried out with input and output intensive, compute intensive and multiprocessed applications leading to an average performance speed up of about 1,000 times and average timing estimation accuracy of less than 3%, when compared with a reference platform based on ISS.

**Keywords:** Design space exploration. Model validation and analysis. Hardware-dependent Software (HdS). Virtual prototyping. Multiprocessing architectures.

# *Lista de Figuras*

# *Lista de Tabelas*

# *Sumário*

# 1 Introdução

Os sistemas computacionais têm estado cada vez mais presentes no cotidiano das pessoas, assumindo as mais diversas funções que possibilitam novas e incríveis aplicações. Nos primórdios desta revolução tecnológica, o computador pessoal foi o sistema de propósito geral que obteve maior difusão, atingindo mais de 1 bilhão de unidades produzidas no ano de 2008 e com uma estimativa de superar um total de 2 bilhões de unidades em 2014 (14).

Apesar do sucesso das últimas décadas, o mercado de computadores pessoais está encolhendo, obtendo taxas de crescimento negativo de 19%, em levantamento realizado na Europa ocidental no segundo trimestre de 2011 (15). Esta redução é consequência direta dos equipamentos eletrônicos de consumo que são baseados em sistemas embarcados de propósito específico que vêm assumindo funções antes exclusivas dos computadores pessoais. Considerando somente os dispositivos com interface de rede sem fio, a taxa de crescimento obtida no ano de 2011 foi de 35% com mais de 1 bilhão de unidades fabricadas e com projeção de mais de 2 bilhões em 2015 (16).

A popularização de novos dispositivos de comunicação, como smartphones e tablets, está gerando uma imensa demanda por mais funcionalidades e capacidade de processamento. Todas estas novas funções vêm tornando difícil a distinção entre estes sistemas embarcados e outros sistemas computacionais de propósito geral, como o computador pessoal. Os principais requisitos destes tipos de dispositivos são a eficiência energética, devido a portabilidade e funcionamento baseado em bateria, a presença de interfaces de comunicação sem fio e capacidades de processamento multimídia (12).

Para atender estes requisitos é necessário um conjunto de elementos de processamento (PEs) de propósito específico para executar as funções do sistema com baixo consumo de potência e alto desempenho de processa-

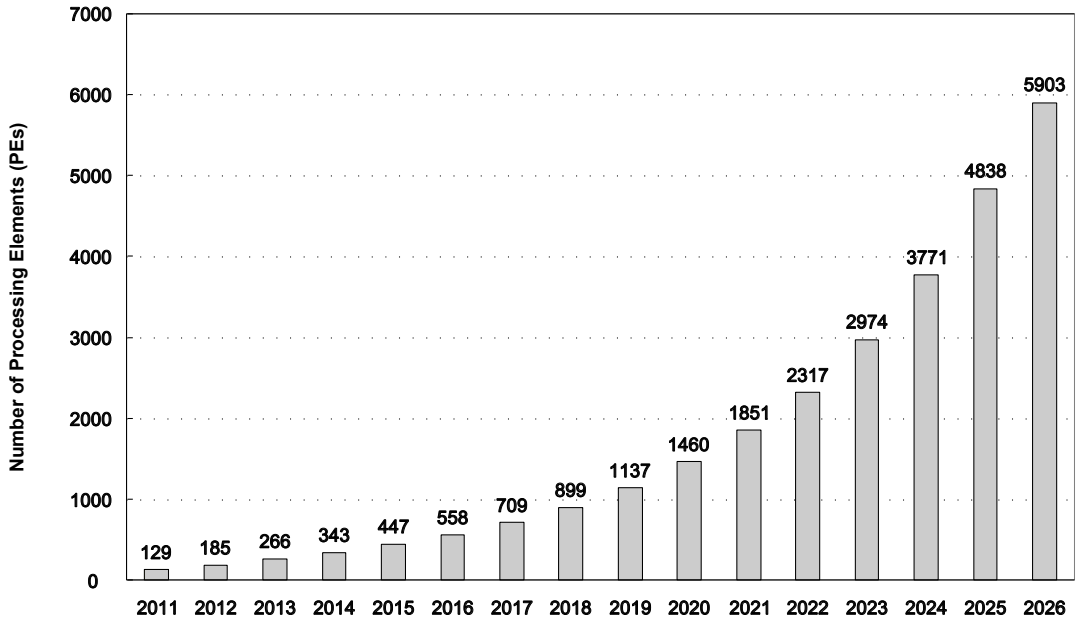


Figura 1: Tendência do número de elementos de processamento (12)

mento. Na figura 1 pode ser vista uma projeção de crescimento exponencial da quantidade de PEs necessária para atender a demanda por desempenho (12). O desafio é não comprometer a eficiência energética com esta tendência de crescimento.

Como não é eficiente a criação de componentes de hardware completamente especializados para atender cada nova funcionalidade do sistema, os elementos de processamento são desenvolvidos para serem reutilizáveis, reduzindo o custo de projeto e o prazo de desenvolvimento. O paradigma de projeto baseado em plataforma (17) procura atender estes requisitos através da flexibilização das implementações de sistemas, absorvendo mais facilmente as constantes atualizações das especificações dos sistemas e amortizando os custos de projeto com reutilização dos componentes.

A principal consequência do paradigma de projeto baseado em plataforma é o aumento da quantidade de software na implementação dos sistemas, basicamente para fornecer a flexibilidade e agilidade necessária na implementação das funcionalidades. Entretanto, devido às limitações de consumo de potência e capacidade de processamento, o software precisa interagir mais diretamente com os componentes da plataforma. Esta maior interação torna o software mais complexo e difícil de ser desenvolvido, demandando maior número de projetistas de software e sendo responsável por pelo menos 80% do custo de desenvolvimento do sistema (18, 13).

## 1.1 Motivação

A crescente discrepância entre o avanço de utilização de software nos sistemas e o insuficiente índice de melhoria de produtividade dos fluxos de desenvolvimento é a principal motivação para a proposição deste trabalho. As seções a seguir irão definir claramente qual o problema que será explorado, destacando as principais dificuldades existentes no estado da arte, além de fornecer informações baseadas em estudos e projeções que ratificam a necessidade por melhores técnicas para o desenvolvimento de hardware e software integrados.

### 1.1.1 Definição do Problema

O fato é que toda esta grande quantidade de funcionalidades e de interfaces fazem uma imensa pressão sobre o fluxo tradicional de desenvolvimento de sistemas baseados em simuladores de conjunto de instrução, principalmente com relação ao software contido neles. A razão para isto é que o desenvolvimento de software possui menor custo e maior flexibilidade quando comparado a uma implementação equivalente em hardware, demandando assim menos tempo para ser lançado no mercado (12). O problema é que o uso massivo de software em sistemas tem trazido gargalos no desenvolvimento para os projetistas de sistemas, devido à incapacidade de atender a crescente demanda por novas funcionalidades das aplicações e aos sistemas serem desenvolvidos com uma taxa de produtividade do hardware e software integrados inferior à capacidade de processamento do hardware (13).

Em um trabalho sobre sistemas embarcados (19), as limitações no desenvolvimento de sistemas foram o tema central e vários pontos críticos foram listados para discussão, como:

- As plataformas estão muito complexas, com processadores heterogêneos (diferentes arquiteturas) e complicada estrutura de memória e interconexões;
- Múltiplas configurações de plataformas que levam a uma explosão no número de versões que precisam ser desenvolvidas, mantidas e portadas;

- O aumento da capacidade de processamento está tornando os processadores complicados e difíceis de programar;
- Os fabricantes de semicondutores não conseguem obter o retorno do investimento no desenvolvimento de software para seus dispositivos;
- É difícil integrar um ambiente de desenvolvimento contendo modelo de plataforma, camada de abstração de hardware ou Hardware Abstraction Layer (HAL) e ferramentas de desenvolvimento;
- Alto desempenho de simulação é essencial para permitir o desenvolvimento eficiente de sistemas complexos.

Aprofundando um pouco mais esta análise sobre o desenvolvimento de sistemas, mais especificamente do software dependente do hardware (HdS) (20), é possível obter mais alguns dados sobre sua complexidade e criticidade. Em um estudo realizado sobre gerenciadores de dispositivos de hardware (device drivers) foi constatado que 70% (21) do código fonte base do Linux (22) é dedicado a controlar periféricos, como unidades de armazenamento, interfaces de teclado e dispositivos USB.

Como os sistemas tem se tornado cada vez mais complexos, a utilização de sistemas operacionais tem sido cada vez mais um requisito obrigatório em sistemas embarcados, para permitir que todas estas funcionalidades sejam adequadamente escalonadas e gerenciadas. Outro aspecto do uso de SO é a necessidade de permitir que futuras atualizações, tanto de hardware como de software, do sistema possam ser feitas com um menor esforço de projeto.

O crescente aumento da complexidade dos sistemas demandou a utilização de sistemas operacionais para gerenciar os recursos, suportar a multiprogramação e controlar os dispositivos de hardware através do HdS. Como software dependente do hardware possui grande representatividade no código fonte base do Linux (21), tem-se como consequência que a maioria das falhas nos sistemas é decorrente de problemas em sua implementação.

Por tudo que foi exposto, é importante a proposição de mecanismos e de técnicas que permitam melhorias no desenvolvimento de sistemas, com suporte ao desenvolvimento e validação eficientes de software integrado com hardware. Estas melhorias devem impactar diretamente na qualidade do sistema, reduzindo o número de falhas através do desenvolvimento em um nível

mais alto de abstração e com uma maior produtividade no fluxo de projeto destes sistemas.

### 1.1.2 Projeções e Tendências

O projeto de sistemas baseados em plataformas virtuais é considerado a principal base para construção das inovações necessárias para resolver ou minimizar os gargalos no desenvolvimento de software dependente do hardware (19), utilizando como estratégias de desenvolvimento:

- Tornar o desenvolvimento do hardware e software mais integrado, onde os fluxos de projeto de hardware e software estão conectados. Esta estratégia tende a evitar que riscos de alto impacto no sistema sejam identificados tardiamente e causem efeito negativo nos custos e cronograma do projeto;
- Suportar a verificação conjunta do hardware e do software, garantindo a melhor qualidade e cobertura dos testes realizados. Esta verificação é feita utilizando modelos detalhados de simulação que permitam ao projetista analisar o comportamento dos componentes de hardware e software integrados.

As plataformas virtuais já são utilizadas tanto pela indústria como pela academia, podendo ser citado o padrão IEEE 1685 (23) que define uma interface para empacotamento, integração e reutilização de componentes eletrônicos. A grande inovação necessária nos fluxos de desenvolvimento consiste na melhoria ou substituição dos modelos em nível de instrução (ISS), uma vez que estes modelos baseados em instruções apresentam um dos principais gargalos de desempenho nas simulações de sistemas. Apesar deste baixo desempenho de simulação, principalmente em sistemas complexos, os resultados obtidos possuem uma alta precisão que é fundamental para o desenvolvimento de hardware e software integrados. Por isto, as principais estratégias propostas buscam um modelo integrado de desenvolvimento, como o oferecido pelo ISS que possui alta precisão, mas sem sacrificar o desempenho que possui impacto direto na produtividade no desenvolvimento do software (13).

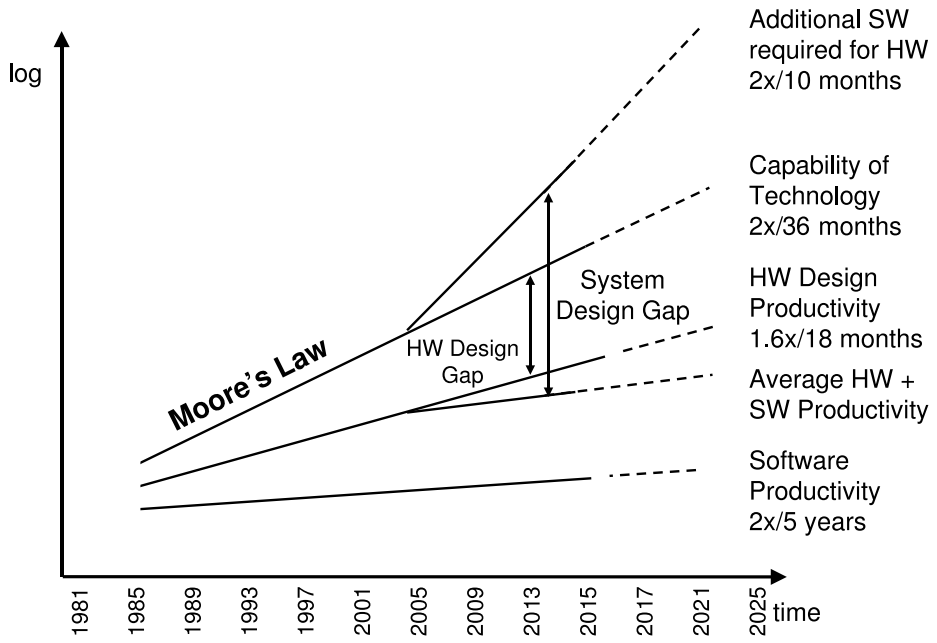


Figura 2: Lacuna de produtividade de sistema (13)

Estas limitações no desenvolvimento e simulação de hardware e software integrados podem ser visualizadas através da lacuna de produtividade de sistema (System Design Gap) (13), ilustrada na figura 2. Estas linhas de crescimento estão baseadas na capacidade de integração da tecnologia, prevista pela lei de Moore (24) que durante os últimos 40 anos estimou com sucesso a tendência de duplicar a quantidade de transistores em circuitos integrados a cada 18 meses.

Apesar das barreiras tecnológicas da fabricação de semicondutores, esta mesma taxa de melhoria na capacidade de processamento é alcançada com o dobro do tempo. Sobre esta linha de referência podem ser traçados diversos outros aspectos diretamente relacionados com esta melhoria de processamento e ligados ao desenvolvimento de software, como está descrito a seguir:

- Para controlar o hardware disponível seria necessário que a quantidade de software para este propósito duplicasse a cada 10 meses. Este tipo de software é dependente do hardware (HdS), ou seja, é especificamente desenvolvido para aqueles novos recursos de hardware;

- A construção de sistemas (hardware e software) possui produtividade média inferior ao avanço da tecnologia de construção do hardware, gerando um subaproveitamento dos recursos disponíveis (13);

- As melhorias na implementação de software só conseguem duplicar sua produtividade a cada 5 anos (13), sem considerar o impacto do hardware neste processo.

Neste contexto, é interessante observar a crescente demanda por software para interagir com o novo hardware desenvolvido e ao mesmo tempo a produtividade do software e do hardware integrados está em uma taxa de crescimento inferior à produtividade do hardware. A consequência direta deste cenário é uma crescente lacuna entre a demanda por software e a melhoria de produtividade em seu desenvolvimento, principalmente em um contexto de integração com o hardware.

## 1.2 Objetivos do Trabalho

Este trabalho tem como objetivo principal o desenvolvimento de mecanismos de modelagem de plataformas em alto nível de abstração que suportem simulações mais eficientes através da execução nativa, operando em nível de transação e suportando, através da modelagem de interfaces e de estimativa de tempo simulado, o desenvolvimento de software dependente de hardware.

Para satisfazer adequadamente o desenvolvimento de sistemas complexos com uma crescente quantidade de periféricos, os projetos de plataformas baseados em processadores vêm conquistando espaço por sua flexibilidade de implementação e pelo aumento de produtividade (13). Entretanto, o aumento na proporção de software utilizado nos sistemas cria uma demanda por técnicas de desenvolvimento e de simulação que sejam mais eficientes e que forneçam estimativas mais precisas sobre o comportamento do sistema.

A modelagem proposta procura preencher as lacunas identificadas na pesquisa do estado da arte, superando os desafios inerentes dos principais problemas em aberto e atendendo a um conjunto de requisitos elicitados para sua implementação.

### 1.2.1 Lacunas do Estado da Arte

Durante a realização da pesquisa de trabalhos relacionados no estado da arte, detalhada no capítulo 3, foram identificadas lacunas nas abordagens propostas, que trazem oportunidades de melhorias. Mesmo algumas destas lacunas estando preenchidas em alguns trabalhos, outros problemas continuam em aberto, gerando soluções que não atendem completamente a todos os objetivos. A descrição destas lacunas é feita nas subseções a seguir, com descrições detalhadas e exemplos do cenário de aplicação.

#### 1.2.1.1 Simulação Nativa da Implementação do Software

Consiste em utilizar a implementação em código fonte do software para a realização de simulações e análises do sistema, sendo este compilado para o próprio sistema de desenvolvimento. Na busca por componentes com alto nível de abstração, alguns dos trabalhos considerados no estudo do estado da arte (25, 10, 11) tem recorrido a modelos de alto nível de sistema para sua especificação, ao invés de utilizar uma implementação em linguagem de programação. Além de demandar novas habilidades da equipe de projeto, estes modelos necessitam de uma infraestrutura adicional para serem simulados e para que seja feita sua conversão em código fonte para compilação.

Como é importante avaliar a própria implementação do sistema especificado em linguagem de programação, ao invés de utilizar modelos de alto nível do sistema, a corrente de trabalhos de execução nativa (2, 3, 4, 5) utiliza a própria implementação baseada no código fonte do software para realização de simulações no ambiente de desenvolvimento. O código fonte do software é instrumentado com informações sobre a arquitetura alvo, permitindo a geração de modelos de sistema com alto nível de abstração e exibindo um maior desempenho devido a execução nativa.

Apesar de alguns trabalhos suportarem a implementação do software em linguagem de programação para execução em um ambiente nativo, a principal lacuna existente nos trabalhos está na necessidade de anotação estática do código fonte do software com informações de tempo simulado de uma determinada arquitetura alvo. Isto quer dizer que os trabalhos que utilizam a implementação do software em código fonte demandam informações pre-

cisas sobre a execução do sistema em um modelo detalhado, recaindo nos gargalos associados a estes modelos, como o baixo desempenho de simulação e a alta complexidade de desenvolvimento deste modelo.

#### 1.2.1.2 Dependência da Arquitetura Alvo

Já foi visto que a adoção de um modelo de alto nível para representar o software (25, 10, 11) possui vantagens e desvantagens, entretanto, por normalmente trabalhar em um nível de abstração mais elevado, não possui uma dependência obrigatória de uma determinada arquitetura alvo. Podem ser utilizadas especificações independentes de tecnologia que não possuem nenhuma relação direta com nenhuma arquitetura de processador, focando nos aspectos comportamentais e não em sua implementação, como largura de banda ou quantidade de núcleos de processamento. Esta característica permite ao projetista realizar uma análise do sistema sem que decisões prematuras de projeto precisem ser tomadas, como a escolha da arquitetura do processador ou do barramento de interconexão.

Quando a implementação em linguagem de programação do software é utilizada (2, 3, 4, 5), é preciso que as informações sobre sua execução em uma arquitetura alvo definida sejam conhecidas. Esta necessidade implica na escolha e na disponibilidade de um modelo preciso da arquitetura alvo escolhida, ou seja, a avaliação do sistema está sendo atrelada a escolha de uma tecnologia de processamento. O cenário ideal é avaliar o sistema de forma independente de tecnologia, verificar as restrições e características com simulações e, com estas informações, decidir qual arquitetura é mais adequada para ser utilizada.

É importante que a própria implementação do software seja avaliada, com precisão e alto desempenho, mas é interessante que a definição da arquitetura alvo possa ser postergada. Assim, o projetista é capaz de avaliar o sistema com maior flexibilidade e menor dependência tecnológica. Assim, a escolha de arquitetura alvo é feita de forma mais adequada, baseando-se em informações coletadas durante a análise do comportamento do sistema.

#### 1.2.1.3 Reutilização de Componentes da Plataforma

Em todas as abordagens pesquisadas do estado da arte foi observado que grande parte dos componentes da plataforma estão acoplados à aplicação. Isto significa dizer que caso a aplicação seja alterada, é preciso modificar os modelos de alto nível do sistema (25, 9, 10, 11), realizar um novo particionamento do software quando se utilizam abordagens híbridas (7, 8) ou realizar novas simulações em modelos precisos e de baixo desempenho para obtenção de informações de execução na plataforma alvo (2, 3, 4, 5).

É muito importante que os componentes de hardware e de software desenvolvidos possam ser reutilizados em diferentes aplicações, sem necessidade de retrabalho. É desejável obter as mesmas características dos modelos de processadores baseados em instruções que são capazes de executar qualquer aplicação binária compilada para a arquitetura alvo suportada.

A lacuna identificada está no baixo índice de componentes reusáveis nos trabalhos considerados no estado da arte, que se for atendida irá permitir que diferentes plataformas e aplicações reutilizassem grande parte dos componentes já desenvolvidos, reduzindo o esforço de projeto.

### 1.2.2 Desafios de Implementação

Para preencher todas as lacunas identificadas nos trabalhos considerados no estado da arte, é preciso entender quais são os desafios inerentes a estes problemas e propor soluções para superar estes obstáculos, que estão listados a seguir:

- Utilização de ferramentas nativas: diferentes plataformas possuem diferentes arranjos de componentes, com diferentes ferramentas, normalmente fornecidas pelos próprios fornecedores destas tecnologias. A grande questão é que já existem e estão amplamente disponíveis diversas ferramentas para desenvolvimento de software em ambiente nativo, ou seja, no sistema de desenvolvimento. Estas ferramentas nativas poderiam ser utilizadas com sucesso para o desenvolvimento de software embarcado, desde que uma descrição executável da plataforma alvo esteja disponível. A modelagem proposta deve ser aplicada aproveitando estes recur-

sos, pois além de tornar o desenvolvimento mais simples e rápido, também irá permitir um uso mais eficiente dos recursos computacionais;

- Alto nível de abstração para modelagem da plataforma: as principais soluções existentes se baseiam no uso de modelos de simuladores de instruções que são precisos, mas são muito complexos para serem construídos e oferecem baixo desempenho de simulação. Por este motivo, é fundamental utilizar modelos com maior nível de abstração (13), menor complexidade de construção e de fácil utilização pela equipe de desenvolvimento;

- Aproximação de tempo simulado precisa: apesar de alto nível de abstração, este trabalho se propõe a fornecer uma modelagem de tempo simulado parametrizável com precisão conhecida e com erro mensurável, de forma que permita aos projetistas estimativas rápidas sobre o comportamento do sistema na plataforma. Com esta modelagem precisa do tempo simulado é esperado que as aplicações executassem sobre um ambiente muito próximo do encontrado em modelos precisos (13), permitindo que as funcionalidades sejam simuladas e avaliadas de forma bem mais rápida;

- Alto desempenho de simulação: é sem dúvida uma das principais características da modelagem proposta, como forma de reduzir os gargalos inerentes ao uso de modelos baseados em conjunto de instruções, permitindo que sistemas complexos possam ser mais rapidamente simulados (13). Esta melhoria é impulsionada pela adoção da estratégia de simulação nativa, onde o software é executado no próprio sistema de desenvolvimento, ao invés de ser emulado em modelos de processadores da arquitetura alvo. Assim, devido a quantidade de reduzida de detalhes que precisam ser simulados, a simulação da plataforma tem seu desempenho melhorado significativamente.

### 1.2.3 Requisitos da Modelagem Proposta

Tendo em mente as lacunas do estado da arte e os desafios de suas implementações, são descritos um conjunto de requisitos para a modelagem dos componentes de hardware e de software do sistema para a realização das si-

mulações. O objetivo desta definição de requisitos é tornar mais sistemática a verificação do preenchimento das lacunas identificadas e da superação dos desafios detectados neste trabalho.

#### 1.2.3.1 Requisitos de Hardware

Os requisitos de hardware são referentes aos componentes da plataforma, como processador, interface com barramento de comunicação e acesso aos periféricos. Para fomentar o suporte necessário da modelagem proposta, considerando as lacunas do estado da arte e seus desafios, são listados e descritos os seguintes requisitos:

- Especificação do processador: deve ser capaz de especificar todos os aspectos relevantes para sua modelagem, como frequência de operação de relógio, taxa de execução de instruções, mecanismo de tratamento de interrupções e integração com os dispositivos da plataforma. Apesar destas descrições estruturais e comportamentais, o sistema pode ser construído em um nível mais alto de abstração independente das arquiteturas alvo de processadores, para permitir maior flexibilidade no projeto do sistema. Este componente deve ter suas interfaces bem definidas e não possuir dependências da aplicação, como ocorre com a anotação de tempo, permitindo sua reutilização em vários contextos;

- Suporte para comunicação externa: permitir que as interfaces de comunicação externa entre os componentes de hardware e de software da plataforma na modelagem proposta possam ser implementadas em diferentes níveis de abstração, como o nível TLM (26) que atende a padrões de interoperabilidade. Desta forma, é permitida uma maior abstração da implementação e um maior desempenho de simulação, sem inviabilizar ajustes no detalhamento do comportamento ou da granularidade das operações;

- Gerenciamento de interrupção: durante a execução do software neste modelo é fundamental, principalmente no contexto de HdS, que a execução da aplicação possa ser interrompida para tratar eventos externos gerados por dispositivos da plataforma eficientemente (27). É um dos

pontos mais relevantes da modelagem proposta, permitindo que o comportamento do sistema na ocorrência de eventos de interrupção sejam observados e analisados, em alto nível de abstração, precisão e desempenho de simulação.

#### 1.2.3.2 Requisitos de Software

Contemplando os componentes de software da plataforma, é definido uma série de requisitos do software, considerando as lacunas e desafios do estado da arte, para que esta modelagem proposta atenda aos objetivos do trabalho, que são:

- Execução nativa da implementação do software: o código fonte da aplicação utilizado na modelagem proposta deve ser o mesmo que será compilado e executado em modelos baseados em conjunto de instruções ou no próprio hardware físico. É desejada a criação de uma modelagem que seja capaz de executar nativamente o próprio software da aplicação, sem utilização de modelos de alto nível que o representem. Este requisito é importante para que o projetista do sistema seja capaz de verificar o comportamento e o desempenho da implementação do software em uma plataforma com um melhor compromisso entre o desempenho de simulação e precisão das estimativas nos resultados obtidos (5);

- Desenvolvimento de HdS: é a parte do software que interage diretamente com o hardware, sendo crítico para o funcionamento dos sistemas. Por isto, seu desenvolvimento deve ser completamente suportado através de uma interface de software baseada em registradores, de forma transparente e com o mesmo comportamento para o programador da aplicação. Desta forma, é esperado que o desenvolvimento completo de camadas de abstração de hardware (HAL) e de gerenciadores de dispositivos (device drivers) seja suportado de forma mais eficiente;

- Suporte para preempção: as rotinas de tratamento de interrupção precisam ser suportadas para tratar em software os eventos gerados pela plataforma, permitindo seu correto funcionamento. Esta preempção do

fluxo de execução do software deve ocorrer como em um modelo baseados em conjunto de instruções ou na implementação física, permitindo que os eventos sejam tratados com o máximo de precisão e desempenho. Este recurso é fundamental para suportar operações de entrada e saída mais eficientes, como as baseadas em interrupção ou por acesso direto à memória (27);

- Interface para sistema operacional: a modelagem proposta deve permitir que a aplicação faça uso das interfaces de programação do SO nativo para realização de todas as operações necessárias. Deve ser suportada a adoção de SO com interfaces de programação distintas do SO nativo, devendo ser suportadas por bibliotecas compiladas nativamente ou por modelos que realizem o mapeamento destas funcionalidades no SO nativo. Desta maneira, é esperado que não fossem impostas limitações para a escolha do SO utilizado pelo sistema.

## 1.3 Contribuições para o Estado da Arte

Ao longo das primeiras páginas deste trabalho foi investido um tempo para introduzir conceitos, motivar o leitor para os problemas em aberto e estabelecer uma série de objetivos que a abordagem proposta se propõe a atingir. Nesta seção, as principais contribuições deste trabalho para o estado da arte são listadas:

- Simulação nativa: a modelagem proposta se dedica a criação de modelos de sistemas que possam ser simulados nativamente, ou seja, utilizando diretamente os recursos da plataforma de desenvolvimento. Para tanto, são propostos um conjunto de mecanismos para modelar os aspectos estruturais e comportamentais da arquitetura alvo, em alto nível de abstração, utilizando todos os recursos de ferramentas disponíveis nativamente;

- Modelagem independente de arquitetura: os modelos de sistemas podem ser criados sem nenhuma dependência de arquiteturas existentes, implementando as funcionalidades do sistema de forma abstrata. A grande vantagem desta característica é permitir uma exploração de espaço de

projeto com maior nível de abstração e sem a determinação prematura de qual arquitetura de processador deve ser utilizada para atender os requisitos do sistema;

- Suporte para desenvolvimento de HdS: como o desenvolvimento deste software é crítico em sistemas (21), foram propostos um conjunto de mecanismos de modelagem para registradores de dispositivos, ponteiros para endereçamento da memória e de rotinas de tratamento de interrupção. Todas as técnicas propostas buscam oferecer um ambiente de desenvolvimento que não implique em mudanças no código fonte da aplicação, através do uso de cabeçalhos reutilizáveis com definições da plataforma. Outra característica importante é permitir que o desenvolvimento de software seja realizado em estágios mais iniciais do projeto;

- Estimativas estáticas do sistema: é proposto um modelo que utiliza parâmetros de configuração para realização de estimativas de tempo simulado do sistema. A escolha destes parâmetros depende exclusivamente dos componentes da plataforma e de sua aplicação. Após o processo de calibração deste modelo, com necessidade de poucas simulações, é possível estimar estaticamente o comportamento temporal do sistema para diferentes configurações de parâmetros, sem a necessidade de simulações adicionais. Com esta ferramenta de análise, o projetista do sistema é capaz de obter previsões sobre o comportamento do sistema em um amplo espaço de projeto e de definir que valores de parâmetros melhor atende aos seus objetivos;

- Suporte para multiprocessamento: o aumento exponencial da complexidade e do número de funcionalidades implementadas, os projetistas de sistema encontram em plataformas multiprocessadas o caminho para atender as demandas de projeto. Este trabalho é desenvolvido para suportar a simulação de sistemas com múltiplos processadores, além de toda a interface de programação necessária para escalonamento, sincronização e coerência dos processos. Entretanto, ainda não é suportada a modelagem de cache de dados e de instrução para execução nativa do software, sendo necessária a utilização de memória compartilhada não mapeada em cache para comunicação entre os processadores.

## 1.4 Estrutura do Documento

Para facilitar a navegação e melhor entendimento, este documento está estruturado em capítulos e seções, que são:

- Capítulo 1 - Introdução: fornece informações básicas sobre a área deste trabalho, com motivações que justificam a relevância dos requisitos elicitados e por fim os objetivos que esta abordagem se propõe a atingir;
- Capítulo 2 - Conceitos Básicos: capítulo dedicado a explicação de conceitos básicos do trabalho, melhorando o entendimento do leitor e gerando maior familiaridade com o tema abordado. Esta parte é de fundamental importância para o público não especialista nesta área de conhecimento;
- Capítulo 3 - Estado da Arte: pesquisa vasta e detalhada, considerando um período de tempo de quase 10 anos, para se observar os caminhos e abordagens que vêm sendo desenvolvidas nesta área, além de demonstrar que os problemas que foram propostos são relevantes e ainda possuem pendências que os trabalhos considerados no estado da arte ainda não resolveram adequadamente;
- Capítulo 4 - Modelagem Proposta: descrição detalhada da modelagem proposta, com todas as especificações e considerações, permitindo que o leitor perceba todos os conceitos e seja capaz, inclusive, de criar provas de conceito deste trabalho;
- Capítulo 5 - Estimativa de Tempo Simulado: neste capítulo, as técnicas e mecanismos propostos para estimativas de tempo simulado na plataforma virtual são detalhados, descrevendo como o tempo de execução nativo do software é mapeado, realizando análises experimentais e destacando o suporte para multiprocessamento;
- Capítulo 6 - Resultados: os resultados gerados estão localizados neste capítulo, sistematizando, através de tabelas e gráficos, todos os dados obtidos, sempre os comparando com modelos precisos de ISS para análise de erro e de melhoria de desempenho;

- Capítulo 7 - Conclusão: os conceitos e resultados obtidos já foram colocados à prova, e neste capítulo uma revisão de tudo que foi realizado é feita, concluindo como os objetivos foram satisfeitos e vislumbrando possibilidades para trabalhos futuros e melhorias.

# 2 Conceitos Básicos

Este capítulo é dedicado a explicitar os conceitos básicos mais importantes para o melhor entendimento deste trabalho, principalmente quando o leitor não for especialista na área de projeto e modelagem de sistemas digitais. Todos estes conceitos estão estruturados nas seções a seguir, sempre tendo como título o nome do conceito que está sendo explicado.

## 2.1 Linguagem de Projeto em Nível de Sistema

A modelagem dos componentes de uma plataforma de um sistema digital demanda uma linguagem de projeto em nível de sistema ou System Level Design Language (SLDL) (28). Visando suportar os diversos aspectos do sistema, são definidos uma série de elementos estruturais (módulos, portas e canais) e comportamentais (processos, eventos e tempo simulado). As principais SLDL utilizadas são SpecC (29, 30) e SystemC (31, 32) que são baseadas nas linguagens de programação C e C++, com os principais objetivos de:

- Modelagem do sistema: com os construtores disponíveis, é possível modelar componentes de hardware e software para a construção de uma especificação executável do sistema. Estes componentes podem ser descritos e conectados, com diferentes níveis de detalhes em seu funcionamento e comunicação com outros componentes. A principal contribuição de SpecC e SystemC reside em se basear em linguagens de programação convencionais (C e C++) para especificar as descrições estruturais e comportamentais dos componentes do sistema;
- Exploração de arquitetura: como existem várias arquiteturas possíveis para um sistema, a exploração arquitetural é fundamental para se avaliar qual é a melhor arquitetura para implementação do sistema. Quando se fala

em definir a melhor arquitetura, está sendo avaliado quantos processadores devem ser utilizados, os seus tipos e capacidades de processamento. O impacto destas escolhas recai diretamente sobre as principais métricas de um projeto de sistemas: desempenho, área e potência consumida;

- Análise de desempenho: as decisões arquiteturais possuem diversos aspectos que precisam ser avaliados, como a quantidade de instruções executadas ou de operações de ponto flutuante realizadas em um determinado espaço de tempo. Este requisito não funcional é mais difícil de ser analisado pelo fato de possuir um comportamento dinâmico, dependente da simulação e dos resultados gerados. Ou seja, sem uma especificação executável ou um ambiente de simulação é praticamente impossível prever o desempenho do sistema até que ele já tenha sido implementado. A utilização de uma SLDL para avaliação do desempenho do sistema em estágios mais iniciais do projeto pode evitar que mudanças mais complexas sejam postergadas para os estágios finais do desenvolvimento, evitando os problemas de atraso de cronograma e aumento de custos;

- Verificação funcional: focando nos aspectos funcionais do sistema, a verificação funcional é responsável por assegurar o comportamento esperado do sistema (33). Como não é possível garantir a ausência de erros, principalmente em sistemas complexos, a verificação funcional tem o foco de garantir que o comportamento especificado seja equivalente ao comportamento modelado ou implementado. Com o suporte de uma SLDL é criado um ambiente de verificação que permite integrar em diversos níveis de abstração os componentes do sistema, como processadores e periféricos, para que sejam gerados estímulos em suas entradas e para que os resultados obtidos em suas saídas sejam analisados;

- Síntese de alto nível: as principais SLDLs foram desenvolvidas com o foco principal de proporcionar construtores para modelagem dos componentes de sistema (30, 32). Entretanto, com advento de novos algoritmos e ferramentas de síntese de alto nível (34, 35), foi viabilizada a geração de componentes de hardware e de software a partir destes modelos com alto nível de abstração. A grande vantagem deste processo está na abstração e automatização de uma série de etapas de desenvolvimento sus-

ceptíveis a erros que em fluxos tradicionais são realizadas manualmente.

As SLDLs são essenciais para concepção e modelagem de sistemas, viabilizando uma análise dos diversos aspectos em um estágio inicial do projeto, proporcionando ao projetista uma ferramenta importante para modelagem, exploração arquitetural, análise de desempenho, verificação funcional e síntese de alto nível. Sem estes recursos, o projeto sempre estaria limitado a simulação de comportamentos implementados através de portas lógicas ou linguagens de transferência de registradores (RTL) (36) que possuem um baixo nível de abstração e um alto nível de complexidade em sua construção.

### 2.1.1 Descrição Estrutural

Sendo parte básica para definição de estruturas em uma SLDL, os módulos, as portas e os canais permitem que os componentes do sistema sejam modelados com modularidade e com interfaces para comunicação com outros componentes da plataforma. Estas interfaces são genericamente chamadas de portas e sempre estão associadas a um módulo, desempenhando um papel de interface de acesso para o módulo. Podendo solicitar informações externas ou responder solicitação, as portas são a maneira pela qual os módulos trocam informações e cooperam entre si. Outro elemento necessário neste processo de comunicação são os canais de comunicação que realizam o transporte dos dados entre as interfaces que conectam e implementam um protocolo para comunicação.

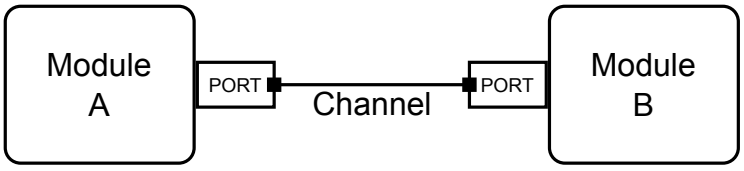


Figura 3: Elementos estruturais de uma SLDL

A figura 3 ilustra como seria um arranjo com dois módulos (Module A e B), cada um com uma porta (Port) e interconectados por um canal de comunicação (Channel). Neste exemplo, os módulos A e B possuem um conjunto de funcionalidades que só podem ser acessados através de requisições em suas portas que precisam estar conectadas através de um canal para permitir o tráfego dos dados.

### 2.1.2 Descrição Comportamental

Em um projeto de sistema é fundamental que o comportamento seja modelado em alto nível de abstração, possibilitando ao projetista representar as funcionalidades e observar seu comportamento de forma rápida. Tudo começa com a definição, dentro de módulos, de processos que irão executar um algoritmo específico, podendo ter seu comportamento ativado de várias maneiras. Provavelmente a principal característica observável na execução de um modelo de sistema em alto nível de abstração seja o tempo simulado, pois confere ao projetista a noção de quanto tempo foi consumido para executar uma determinada função. Sem a noção de tempo simulado o sistema realiza suas tarefas em um tempo nulo ou instantâneo, não permitindo ao projetista avaliar o desempenho atingido ou as latências envolvidas.

Com o estabelecimento da linha de tempo simulado, seja por atrasos anotados no modelo ou pela contagem de ciclos de relógio, é possível especificar de forma mais realista o comportamento dos processos que executam concorrentemente. A concorrência pode ser definida pelo escalonamento de tarefas dentro de um mesmo intervalo de tempo, assim o tempo simulado é muito mais que um contador de quanto tempo foi simulado, permitindo que funções que precisam executar em paralelo sejam avaliadas e facilmente modeladas.

## 2.2 Software dependente de Hardware

O software que depende do hardware ou Hardware-dependent Software (HdS) (20), é um tipo de software de sistema responsável pelo controle e gerenciamento dos componentes de hardware da plataforma. Este controle pode ser utilizado para realização de operações de entrada e saída de dados, gerenciamento de interrupções, assim como para realização de operações de inicialização e de configuração do sistema. Quando bem concebido, este software deve ser bem modularizado, geralmente em associação com uma camada de abstração de hardware, para reduzir o impacto de mudanças na plataforma e evitar dependências na implementação das aplicações do sistema.

O papel do HdS deve ficar mais claro ao ser observado a figura 4, que ilus-

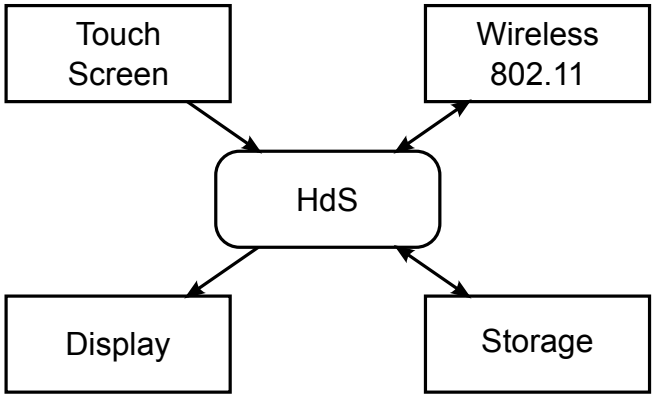


Figura 4: Cenário de uso de HdS

tra as interações com os dispositivos em um sistema de smartphone ou tablet modernos. Neste cenário, o HdS tem a responsabilidade de gerenciar os diversos dispositivos de uma plataforma, como tela sensível ao toque (Display e Touch Screen), redes sem fio do padrão IEEE 802.11 (37) e unidades de armazenamento em disco ou memória (Storage). Com o desenvolvimento padronizado da camada de HdS, a aplicação faz uso de todos estes recursos através de uma interface de acesso ao hardware bem definida, não necessitando de modificações caso algum dispositivo seja atualizado ou substituído, deixando todo o impacto da mudança somente no código fonte do HdS.

### 2.2.1 Gerenciadores de Dispositivos

Mais conhecidos como device drivers, estes componentes de software são os melhores exemplos de HdS. Este tipo de software é muito conhecido pelos usuários de computadores pessoais, em grande parte pela utilização de diversos dispositivos, como impressoras ou placas de vídeo, por exemplo. Até as pessoas leigas sobre o tema percebem que para que um novo dispositivo possa funcionar corretamente em seu computador, é preciso utilizar um software que habilite os serviços do dispositivo para as aplicações do sistema.

A figura 5 ilustra, em termos de camadas de software, onde fica situada a camada dos gerenciadores de dispositivos (Device Driver) (38). Como a aplicação do usuário está construída sobre uma infraestrutura de sistema operacional (OS) e precisa acessar os dispositivos, é preciso que a programação do gerenciador traduza as requisições do sistema operacional (SO) para o dispositivo e vice versa.

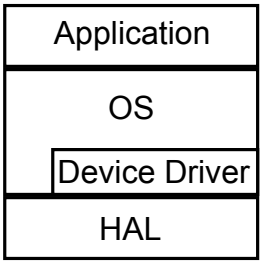


Figura 5: Estrutura de camadas de HdS

Pode ser observada também, no nível mais inferior, a camada de abstração de hardware que é responsável por encapsular tudo que for específico da plataforma das camadas superiores. Este encapsulamento se refere a forma como os recursos do hardware são acessados, pois é possível que o mesmo tipo de recurso possua diferentes configurações de acesso e de gerenciamento. Assim, mesmo que a plataforma seja alterada e o mesmo dispositivo seja utilizado, o gerenciador de dispositivo pode ser utilizado sem modificações e o impacto fica restrito a camada de abstração de hardware que também é um tipo de HdS.

### 2.2.2 Interface de Hardware

Para suportar a grande diversidade de dispositivos e reduzir o impacto na implementação dos componentes de software, principalmente dos sistemas operacionais, foi desenvolvida da camada abstração de hardware ou Hardware Abstraction Layer (HAL). Esta camada de software possibilita ao SO uma interface direta com o hardware da plataforma, resolvendo os problemas referentes a inicialização, configuração e gerenciamento de suas funções. As camadas superiores enxergam uma interface de software alto nível e portável, sem maiores detalhes sobre que tipo de hardware está sendo utilizado na plataforma.

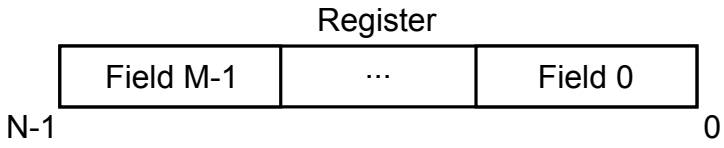


Figura 6: Interface através de registradores

Como o objetivo é entender seu funcionamento, é preciso saber como a HAL implementa a interface direta com o hardware. Isto geralmente é reali-

zado por escritas e leituras em registradores, como está ilustrado na figura 6. Um registrador é definido por um tamanho total de $N-1$ de bits, que normalmente depende da capacidade da plataforma, e uma série de $M-1$ campos de tamanho variáveis, para armazenar diferentes tipos de informações, com diferentes permissões de acesso para leitura e escrita. Ainda existem duas maneiras distintas de ter acesso a estes registradores: na primeira, o processador possui instruções específicas para acessar estes registradores; e na segunda, todo o acesso é feito através de mapeamento destes registradores em endereços de memória.

### 2.2.3 Tipos de Dispositivos

Foi visto como o HdS pode se apresentar em um sistema, seja através de gerenciadores de dispositivos ou de uma camada de abstração de hardware, mas ainda não foi feita uma categorização dos tipos de dispositivos que podem ser utilizados em uma plataforma. Considerando a organização do sistema operacional Linux (38), podem ser consideradas três categorias principais:

- Caractere: correspondem a maioria dos dispositivos, representando 52% do total em linhas de código fonte base do Linux para gerenciamento de dispositivos (21), sendo orientados por fluxo de bytes, ou seja, um envio ou recebimento contínuo de informações. São exemplos deste tipo de dispositivo as interfaces de vídeo, teclado, mouse e sistemas multimídia. Geralmente implementam pelo menos as funções de sistema para abertura (read), fechamento (close), leitura (read) e escrita (write). Apesar da similaridade com arquivos, o acesso a dispositivos deste tipo só permitem acessos sequenciais, sem possibilidade de retrocesso no fluxo, como ocorre em arquivos;

- Bloco: suportam acesso randômico aos blocos do dispositivo, onde o acesso randômico é pela capacidade de acesso direto a informação e um bloco representa uma estrutura com um determinado número de bytes, respondendo por 16% do tamanho do código fonte base do Linux (21). As unidades de armazenamento, como discos rígidos, unidades de memória e interfaces USB, são bons exemplos deste tipo de componen-

tes. A diferença entre um dispositivo de bloco e de caractere consiste no fato do bloco possuir vários bytes de tamanho (geralmente com 512 bytes) e que estes blocos podem ser transferidos simultaneamente;

- Rede: como o nome sugere, são os dispositivos relacionados as interfaces de comunicação de um sistema, suportando fluxos de pacotes e representando 27% do código fonte base do Linux (21). Os pacotes são unidades formatadas de dados que permitem a troca de dados entre os dispositivos de rede, como em interfaces de redes com fio ethernet e sem fio wimax, por exemplo. Não são orientados por fluxo de dados, como em dispositivos de caractere ou de blocos, logo não são mapeados como nós do sistema de arquivo, com funções e interfaces distintas para realização das operações de transmissão de pacotes.

Além destes tipos principais, existe um pequeno conjunto de bibliotecas para suportar interfaces comuns de outras famílias de dispositivos, como teclado e mouse. Apesar da categorização proposta estar atrelada ao contexto do sistema operacional Linux, os conceitos vistos existem em todas as demais plataformas, podendo ter nomes diferentes, mas com o mesmo tipo de comportamento descrito nesta seção.

## 2.3 Plataforma Virtual

O conceito de plataforma virtual em desenvolver um modelo do sistema a partir dos modelos de seus componentes (17, 39), com diferentes níveis de abstração com relação a sua implementação em hardware. A principal motivação de se especificar uma plataforma virtual é a obtenção rápida de um ambiente de simulação que suporte a avaliação das decisões de projeto em um estágio inicial de desenvolvimento.

Na figura 7 é ilustrado um diagrama de blocos de uma plataforma virtual genérica para exemplificar como os componentes podem ser integrados. A infraestrutura para simulação de plataformas virtuais é suportada por uma SLDL que através de seus construtores permite a modelagem e a simulação dos componentes do sistema. Para integrar os componentes da plataforma de forma escalável é utilizado um barramento para interconexão (Bus Inter-

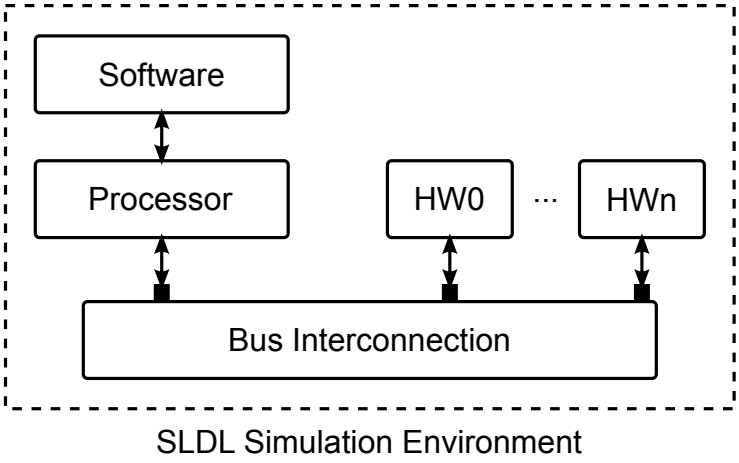


Figura 7: Diagrama de blocos de uma plataforma virtual

connection), utilizando uma interface padronizada que permite a reutilização dos módulos em diferentes plataformas, sem necessidade de modificação de sua implementação. Dentre os componentes de hardware, o processador (Processor) e os dispositivos (HW0 até HWn) representam as funcionalidades reprogramáveis e fixas da plataforma, respectivamente. A capacidade de reprogramação, através da utilização de uma sequência de instruções em formato binário armazenada em memória (Software), confere ao sistema a flexibilidade necessária para ser adequada a diferentes aplicações e permite a atualização de suas funcionalidades. Já os dispositivos são controlados pelo processador, realizando o processamento dedicado e de transferência de dados para operações de entrada e saída.

Durante o processo de desenvolvimento, os componentes podem ser implementados em diversos níveis de abstração, desde um modelo de mais alto nível de abstração, descrevendo apenas aspectos funcionais, até um nível mais baixo de abstração que executa instruções de uma determinada arquitetura de processador, que é uma representação mais próxima do hardware real. Os detalhes de cada um destes níveis serão fornecidos nas subseções a seguir, considerando como exemplo a aplicação produtor e consumidor para facilitar o entendimento dos conceitos descritos.

### 2.3.1 Modelagem em Nível de Aplicação

Na aplicação do produtor e do consumidor existem dois componentes básicos: o produtor, que é responsável pela geração e enfileiramento dos dados,

e o consumidor, que realiza o desenfileiramento dos dados e sua utilização. Tanto a produção como o consumo pode ser realizado concorrentemente, só sendo interrompidos quando o produtor atinge a capacidade máxima de armazenamento de dados em sua fila ou o consumidor já carregou todos os dados disponíveis, estando a fila vazia.

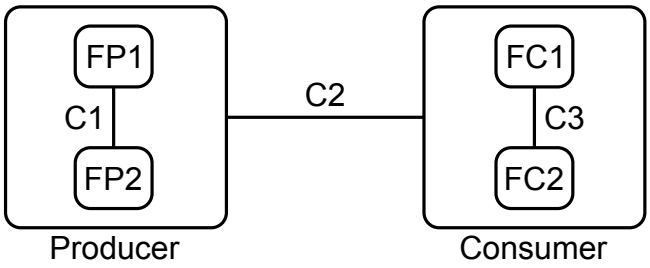


Figura 8: Modelo em nível de aplicação

Na figura 8, os componentes produtor (Producer) e consumidor (Consumer) são módulos do sistema, com suas respectivas funcionalidades e canais de comunicação. No módulo do produtor é observado duas funcionalidades referentes a produção (FP1) e ao enfileiramento dos dados (FP2), através da comunicação interna com o canal C1. Observando o módulo do consumidor, são vistas as funcionalidades de consumo (FC1) e de desenfileiramento (FC2) sendo integradas pela comunicação interna com o canal C3. No nível mais alto da hierarquia, os dois módulos trocam os dados através do canal C2, que neste caso é uma abstração de uma estrutura de fila.

Neste nível de aplicação, o projetista do modelo do sistema só tem a preocupação de refletir os aspectos estruturais e funcionais necessários para que se obtenha uma especificação executável de mais alto nível possível. Todos os recursos necessários para construção do modelo são fornecidos pela SLDL, evitando a necessidade de implementação de estruturas ou algoritmos para comunicação interna e externa, além da organização dos processos concorrentes e do controle de sincronismo.

### 2.3.2 Modelagem em Nível de Tarefas

Sob a perspectiva da modelagem do software do sistema, são criadas tarefas para implementar as funções da aplicação desempenhadas pelo sistema. No nível de tarefas, a concorrência e a comunicação são suportadas por meio de um modelo de sistema operacional (SO) descrito na SLDL. A ob-

tenção de uma descrição no nível de tarefas a partir da especificação em nível de aplicação é obtida pela conversão dos comportamentos descritos em alto nível para uma visão de implementação baseada em SO, possibilitando análises de políticas de escalonamento, sincronização e priorização.

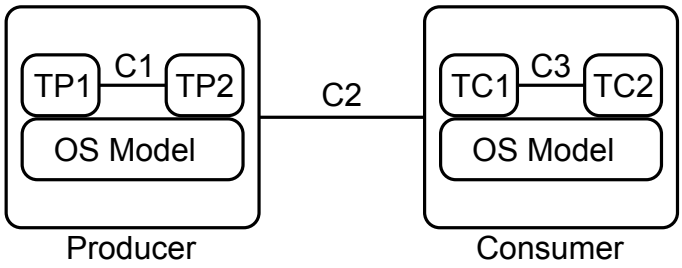


Figura 9: Modelo em nível de tarefas

Na figura 9 é possível ver que foi adicionado um modelo de SO (OS Model) em cada um dos sistemas de produtor (tarefas TP1 e TP2) e de consumidor (tarefas TC1 e TC2). Neste nível de detalhe, o projetista é capaz de avaliar qual o tipo de SO mais adequado e se existe algum tipo de impedimento para que seja obtido o funcionamento esperado do sistema. É interessante ressaltar que por se tratar de um modelo de alto nível, a interface definida pode ser genérica, ou seja, definida de forma abstrata e sem relação com nenhum SO existente, dando ainda mais liberdade ao projetista.

Uma vez definido o SO, diversas configurações podem ser avaliadas dinamicamente, através de simulações, como as políticas de escalonamento das tarefas, mecanismos de sincronização e priorização. Dependendo do tipo de configuração adotada, os resultados obtidos podem ser bem distintos, sendo difícil predizer este comportamento estaticamente. O nível de tarefas é o primeiro momento em que detalhes de implementação do software são explicitados e que o próprio código fonte pode ser utilizado na simulação.

### 2.3.3 Modelagem em Nível de Gerenciamento de Hardware

Como o acesso ao hardware é fundamental, é definido um nível de gerenciamento de hardware (Firmware) que refina ainda mais a interface do software, permitindo o acesso aos dispositivos da plataforma, através de uma camada de abstração de hardware (HAL). Assim como ocorreu no refinamento para tarefas, toda a comunicação externa, que era realizada com uso de

construtores da SLDL, é substituída por uma interface de programação para realização de entrada, saída e controle de interrupção na plataforma.

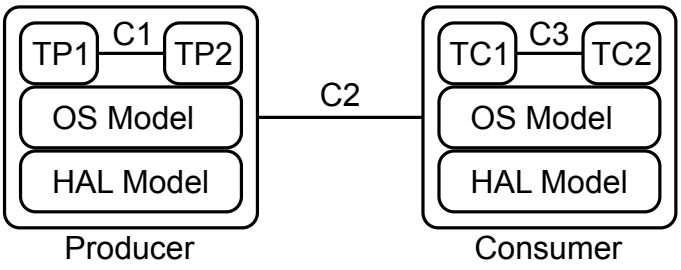


Figura 10: Modelo em nível de gerenciamento de hardware

A figura 10 ilustra bem este novo detalhamento, permitindo que tanto o software do produtor como do consumidor utilizem as funções do modelo de abstração de hardware (HAL Model). Com o refinamento do acesso ao hardware, detalhes sobre o hardware ficam mais evidentes, como os parâmetros de configuração necessários, como taxa de operação ou modo de comunicação, e as características dinâmicas dos dispositivos que estão sendo acessados, podendo trazer ao projetista informações que ajudem na escolha dos dispositivos que melhor se adequem aos requisitos do sistema.

Com a inclusão das interfaces para o SO e HAL, o software está completamente suportado, sob a perspectiva de suas funcionalidades, e já é capaz de ser compilado e executado na plataforma real, desde que a implementação em software do SO e da HAL seja incluída. No contexto de desenvolvimento de HdS, a HAL tem importância primordial por tratar diretamente de todos os aspectos de acesso ao hardware, como processadores e periféricos, devendo também ter seu desenvolvimento suportado por modelos com alto nível de abstração (20, 10) para evitar os longos tempos de execução de sistemas complexos.

### 2.3.4 Modelagem em Nível de Transação

A modelagem em nível de transação ou Transaction-level Modeling (TLM) (26) foi concebida para implementar as transferências de dados nas plataformas com um nível mais alto de abstração e permitir a padronização das interfaces para intercâmbio dos componentes desenvolvidos. Neste paradigma, todos os elementos de comunicação, como portas de entrada e saída, protocolos e a própria topologia do sistema são representados em alto nível de

abstração, evitando detalhes excessivos nas operações e assim favorecendo a descrição dos aspectos funcionais e um melhor desempenho da simulação em comparação com modelagens mais detalhadas.

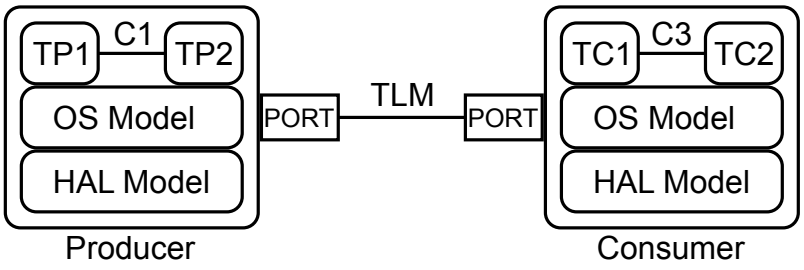


Figura 11: Modelo em nível de transação (TLM)

Neste nível de modelagem, os sistemas do produtor e do consumidor podem ser avaliados considerando as diferentes interfaces disponíveis dos módulos (PORT) e seus protocolos associados nos canais (TLM), como é ilustrado na figura 11. Por exemplo, é possível avaliar se a comunicação será síncrona ou assíncrona, se a interface será serial ou paralela, qual será a largura de banda disponível e o tipo de endereçamento que será utilizado. Todas as operações de comunicação são chamadas de transações, podendo ser iniciadas pelos componentes ativos do sistema, que neste caso é o consumidor, que solicita as informações do produtor.

No nível TLM, o sistema possui pela primeira vez o comportamento, a estrutura (módulos e portas) e a comunicação modelados em alto nível, permitindo ao projetista avaliar todas as escolhas de projeto realizadas. A grande maioria dos modelos de sistema se concentra neste nível, principalmente por apresentar a quantidade de detalhes suficientes para avaliação de grande parte das características do hardware e do software, aliado a um alto desempenho da simulação.

### 2.3.5 Modelagem Funcional de Barramento

Assim como o nível TLM descrito anteriormente, a modelagem funcional de barramento ou Bus Functional Model (BFM) (33) também é um paradigma para criação de modelos de comunicação. Só que ao contrário de TLM, os modelos BFM tem um nível de detalhe muito maior, implementando precisamente as transições dos sinais assim como acontece na implementação real. A grande vantagem desta abordagem é uma visão muito mais detalhada da comuni-

cação real em operação, com todas as informações de temporização e de protocolo. Entretanto, todas estas informações impactam negativamente sobre o desempenho de simulação, pela quantidade gigantesca de detalhes adicionais que precisam ser simulados.

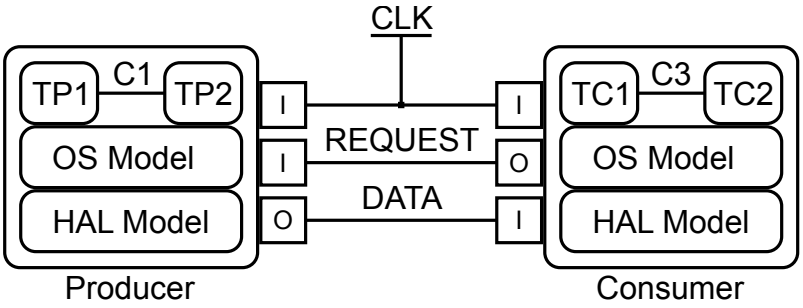


Figura 12: Modelo em nível funcional de barramento (BFM)

No contexto da aplicação de produtor e consumidor, é feito o detalhamento dos sinais envolvidos na realização da troca de informações entre os sistemas, como pode ser visto na figura 12. As portas continuam presentes, mas foram especializadas para lidar com sinais, ao invés de transações, como ocorria no nível TLM. Estas portas podem ser de entrada (I), saída (O) ou ambos os casos (I/O), estando sempre associadas ao tipo e largura de sinal que irão ler ou escrever. Neste exemplo, os sinais são o relógio do sistema (CLK), o sinal de requisição de dados (REQUEST) e o sinal de dados (DATA) para efetuar a transferência dos dados do produtor para o consumidor.

Analisando sob uma perspectiva de caixa preta, não existe distinção entre um modelo de alto nível e uma implementação real, pois, em ambos os casos, o comportamento observado é idêntico. A única diferença é o desempenho superior do modelo BFM e de uma maior flexibilidade, devido ao uso de modelos de alto nível que estão contidos nele.

### 2.3.6 Modelagem em Nível de Instrução

Também conhecidos como simuladores de conjunto de instrução ou Instruction Set Simulator (ISS) (40), estes modelos são provavelmente a forma mais popular de desenvolvimento de sistemas embarcados. O motivo para a adoção frequente deste modelo é o fato de aproximar com grande detalhe o comportamento real do processador, executando o mesmo software em formato binário que irá ser executado na plataforma real.

A excelente precisão obtida nas estimativas de tempo simulado obtidas é decorrente do modelo executar a aplicação instrução por instrução, levando em consideração as otimizações realizadas pelo compilador daquela arquitetura e seu fluxo de execução. Outro aspecto importante é dar ao projetista a certeza que o mesmo comportamento observado no modelo ISS será visto na implementação em hardware, uma vez que em ambos os casos o mesmo código binário é utilizado, sem modelagem ou virtualizações.

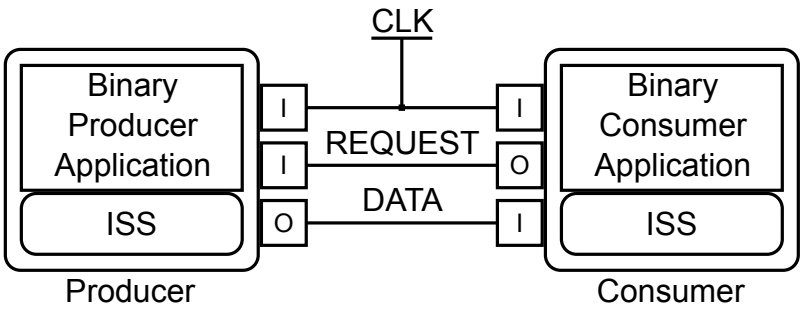


Figura 13: Modelo em nível de instrução (ISS)

Na figura 13 é ilustrada a inclusão do modelo de processador baseado em ISS que executa o comportamento das aplicações binárias de produtor e consumidor, oferecendo o mesmo comportamento que teriam em um processador real. Apesar de toda esta precisão e compatibilidade, a quantidade excessiva de comportamentos que precisam ser inseridos neste modelo ISS o torna extremamente complexo de ser construído e com baixo desempenho de simulação. Um modelo baseado em ISS pode ser descrito para minimizar a complexidade, focando nos aspectos funcionais, ou melhorar a precisão obtida, através da descrição das estruturas internas, como pipeline, podendo ser classificados em: funcional, sem os detalhes internos, como pipeline; e com precisão de ciclo, modelando com detalhes o funcionamento da implementação em hardware.

## 2.4 Arquiteturas Multiprocessadas

Apesar dos grandes avanços tecnológicos, os processos de produção de semicondutores vêm encontrando grandes barreiras tecnológicas decorrentes do tamanho extremamente reduzido dos transistores. Até então, o enfoque de melhoria de desempenho do projeto de processadores foi baseado na execução paralela de instruções ou Instruction Level Parallelism (ILP) (41)

e no aumento da frequência de operação dos circuitos, o que permitiu que uma única unidade de processamento atingisse ganhos significativos de desempenho com o aprimoramento da tecnologia de fabricação.

Estes aprimoramentos da arquitetura do processador e da tecnologia de fabricação dos semicondutores sustentou durante décadas a taxa de crescimento de desempenho e de capacidade de armazenamento dos sistemas, extraindo o paralelismo implícito de uma programação sequencial. Entretanto, para continuar oferecendo ganhos de desempenho a cada nova versão do sistema, a complexidade dos algoritmos de execução paralela das instruções cresce exponencialmente, oferecendo uma melhoria no desempenho de até 40% com a duplicação da complexidade da implementação (27).

Por isto, com a barreira tecnológica da fabricação de semicondutores e a exaustão do paralelismo em um único processador, foi necessária uma mudança de paradigma no desenvolvimento de hardware e software. Esta mudança residiu na paralelização explícita da execução de uma aplicação em software, utilizando mais de um único núcleo de processamento para melhorar o desempenho do sistema. Esta ideia não era realmente inovadora e já estava sendo utilizada em outros contextos a algum tempo, mas sua adoção foi postergada pelas implicações da execução multiprocessada de software (42).

Como o software é uma sequência de instruções, existe uma dependência natural entre estes passos, sendo difícil definir que passos podem ser feitos em paralelo e quais não. Em processadores que exploram o paralelismo em nível de instrução, esta análise é feita em tempo de execução pelo hardware e permite que instruções executem paralelamente, sem interferência do programador. Na exploração do paralelismo em nível de thread, o desenvolvedor de software precisa definir explicitamente como o paralelismo ocorre, através de programação concorrente e mecanismos de sincronismo.

$$Melhoria = \frac{1}{(1 - P) + \frac{P}{N}} \tag{2.1}$$

Para mensurar o ganho de desempenho de sistemas multiprocessados pode ser aplicada a lei de Amdahl (43), descrita pela fórmula 2.1. Por esta formulação, a melhoria de velocidade é limitada pela porção sequencial do software.

Em uma análise ingênua, seria fácil concluir que se existem $N$ processadores ao invés de 1, a velocidade será multiplicada por $N$. Mas, como já foi dito, a melhoria obtida depende exatamente da porção do software que está implementada de forma paralela.

$$\lim_{N\to\infty} Melhoria = \frac{1}{(1-0,95)+\frac{0,95}{N}} = \frac{1}{0,05} = 20 \tag{2.2}$$

Por exemplo, se um determinado programa possui apenas 5% de funções sequenciais ($P = 0,95$) e um número infinito de processadores ($N \to \infty$), é obtida uma melhoria máxima de 20 vezes no desempenho, como pode ser visto nos cálculos realizados na fórmula 2.2. A execução paralela do software pode criar diferentes fluxos de dados e instruções armazenadas, ou seja, como os processadores buscam as instruções da aplicação e como manipulam seus dados, além da função ou papel que os múltiplos processadores desempenham na plataforma (44). Estes pontos serão tema das próximas subseções que detalham a classificação dos diferentes tipos de paralelismo e definem as maneiras como o multiprocessamento pode estar organizado.

### 2.4.1 Classificação do Paralelismo de Processamento

Independentemente da organização de processadores adotada, todos os processadores podem ser caracterizados utilizando o modelo de classificação de Flynn (45). Este modelo classifica todos os sistemas em 4 categorias de acordo com seus fluxos de busca de dados e de instruções:

- Fluxo simples de instrução e dados (SISD): nesta categoria se encontram todas as plataformas uniprocessadas, onde uma única unidade realiza as buscas por instruções que serão executadas e seus respectivos dados que são os argumentos das instruções (46);

- Fluxo simples de instrução e múltiplo de dados (SIMD): uma mesma instrução é executada por múltiplos processadores que operam sobre diferentes fluxos de dados. Estes tipos de sistema exploram o paralelismo em nível de dados, onde cada processador possui seu próprio conjunto de dados e realiza operações gerenciadas por uma unidade central de controle. Este tipo de sistema é muito eficiente em aplicações de multimídia ou pro-

cessamento gráfico que possuem um paralelismo natural na organização dos dados (47);

- Fluxo múltiplo de instrução e simples de dados (MISD): consiste em múltiplos fluxos de instruções e um fluxo simples para busca dos dados. Uma das possíveis aplicações para este tipo de sistema está em aplicações que demandam tolerância a falhas e replicação de tarefas, como em sistemas espaciais (48) ou em reconhecimento de padrões (49), por exemplo;

- Fluxo múltiplo de instrução e de dados (MIMD): cada processador busca seu próprio conjunto de instruções e de dados, focando na exploração do paralelismo em nível de threads. Isto ocorre devido ao fato das threads operarem em paralelo nos processadores, podendo ou não compartilhar informações com outras threads. Este paralelismo oferece maior flexibilidade do que o paralelismo em nível de dados, e por isto, esta categoria é a escolha mais adequada para plataformas multiprocessadas de propósito geral (50, 51).

### 2.4.2 Multiprocessamento Simétrico (SMP)

Neste paradigma de multiprocessamento são adicionados núcleos de processamento idênticos, com mesma interface com a memória principal e os mesmos dispositivos de entrada e saída, com todos os processadores possuindo a mesma função ou papel na plataforma. Este tipo de abordagem é conhecido como multiprocessamento simétrico ou Symmetric Multiprocessing (SMP) (52) e tem sido amplamente adotado em sistemas de propósito geral, como os baseados em processadores AMD (50) e Intel (51), por exemplo.

O funcionamento deste tipo de arquitetura consiste em executar um conjunto de aplicações de forma compartilhada entre todos os processadores, onde cada núcleo é capaz de executar as tarefas que estavam sendo executadas por outro núcleo e vice versa, uma vez que os processos são escalonados em diferentes núcleos. Para garantir a consistência de execução, além de salvar e restaurar o contexto de cada aplicação durante seu escalonamento, as regiões de memória compartilhadas por diferentes processadores tem seu acesso realizado de forma sequencial, para evitar que a ordem de

execução das operações altere o resultado esperado da aplicação.

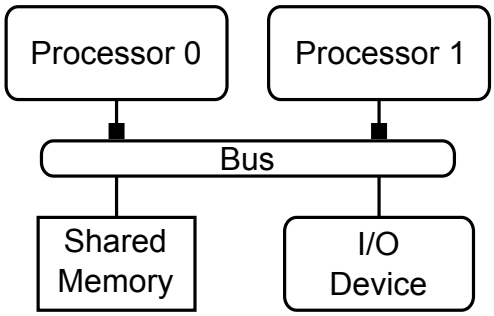


Figura 14: Exemplo de Plataforma SMP

Na figura 14, é exibida uma ilustração de uma plataforma SMP, com dois processadores (Processor 0 e 1), interconectados por um barramento (Bus), além de uma memória principal compartilhada (Shared Memory) e um dispositivo de entrada e saída (I/O Device). Nesta configuração, a única restrição que se aplica ao software é o suporte do sistema operacional, que precisa alocar as tarefas entre os diversos núcleos disponíveis.

Desta maneira, é possível executar um software paralelizado para uma arquitetura uniprocessada em uma arquitetura SMP, necessitando que o sistema operacional suporte os múltiplos processadores da plataforma. Por esta compatibilidade reversa, maior suavidade de transição e facilidade de implementação, em comparação com as plataformas uniprocessadas, a organização SMP tem sido amplamente adotada em sistemas de propósito geral, como os computadores pessoais. Outro aspecto positivo é sua robustez a falhas, pois caso um processador seja desligado para redução de consumo ou falha de funcionamento, o comportamento do sistema pode ser preservado, uma vez que todos os núcleos são equivalentes.

### 2.4.3 Multiprocessamento Assimétrico (AMP)

A melhoria de desempenho do sistema pode ser feita através da adição de processadores heterogêneos ao sistema, com espaço de memória privado para cada processador e a alocação das unidades processamento para executarem funções especializadas. Este conceito é definido na literatura como multiprocessamento assimétrico ou Asymmetric Multiprocessing (AMP) (53), sendo mais utilizado em sistemas dedicados ou de propósito específico. Com o espaço de memória privado, cada processador possui sua

própria memória de programa e seu conjunto isolado de aplicações para ser executado, podendo utilizar uma memória de acesso global para compartilhamento de informações. É possível um cenário de memória centralizada, assim como ocorre no multiprocessamento simétrico, mas os processadores possuem funções ou recursos diferentes, como núcleos com diferentes capacidades de processamento e consumos de potência, como ocorre na plataforma Samsung Exynos 5 Octa (54).

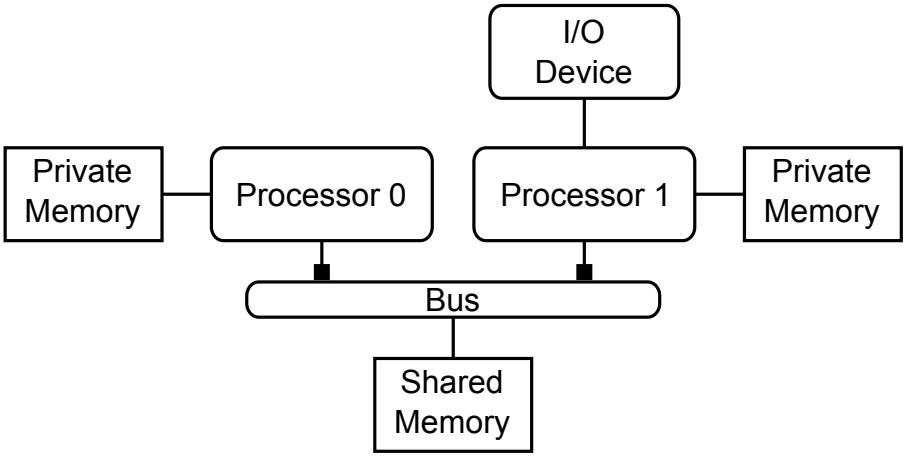


Figura 15: Exemplo de Plataforma AMP

Na figura 15, é fornecido um exemplo de plataforma AMP com dois núcleos de processamento (Processor 0 e Processor 1), interconectados por um barramento (Bus). Cada processador possui sua memória privada (Private Memory), com seu próprio espaço de endereçamento, mas existe uma memória compartilhada (Shared Memory) para troca de informações. Outro aspecto, que também foi ilustrado no exemplo, foram os diferentes recursos disponíveis para cada processador, neste caso o processador de número 1 possui acesso exclusivo a um dispositivo de entrada e saída (I/O Device).

Devido a estas especializações dos processadores, as aplicações que executam em cada núcleo devem ser especialmente criadas para utilizar seus recursos. Assim, é difícil construir uma plataforma assimétrica que tenha um propósito geral, sem impactar no projeto do software que será executado. Outro ponto importante é que em uma situação de falha de algum núcleo, o sistema inteiro pode ser comprometido, uma vez que cada processador possui uma função específica e isolada.

## 2.5 Avaliação de Desempenho

Para se conhecer a capacidade de realização de funções ou o desempenho de um sistema computacional (55), é preciso definir como será feita a medição das operações que precisam ser realizadas, ou seja, definir qual a métrica que será utilizada, e quanto tempo foi consumido para realização destas operações, geralmente expresso em segundos. As técnicas utilizadas para obtenção destas informações são baseadas em modelos aproximados, sistemas de simulação ou medição dos resultados em experimentos com um protótipo do sistema. Seguindo a ordem de enumeração das técnicas citadas, existe um significante incremento no esforço empregado para avaliação do desempenho, entretanto, os resultados obtidos também apresentam uma maior precisão. Para a avaliação quantitativa de desempenho de sistemas podem ser utilizadas diversas métricas, como:

- Taxa de processamento: com esta métrica é possível determinar quanto volume de dados está sendo processado por um sistema. Considerando um sistema de processamento de imagens, a unidade de dados é um quadro de imagem e a taxa de processamento pode ser expressa em termos de quadros processados por segundo (fps) ou da taxa de bits equivalente (bps). Para quem analisa o sistema, cada uma destas unidades é capaz de definir a capacidade do sistema e estabelecer uma comparação com outras implementações;

- Tempo de resposta: através da medição de quanto tempo um sistema leva para realizar uma tarefa é possível verificar o desempenho de sua implementação. Considere o exemplo de um sistema de frenagem de veículos, onde o tempo de resposta de acionamento é essencial para sua boa avaliação. Quanto mais curto o tempo consumido na ativação dos freios, melhor o desempenho do sistema, e esta é uma métrica que pode ser utilizada para comparar diferentes implementações;

- Largura de banda: consiste em medir quanto de capacidade de envio paralelo de informações que um canal de transmissão possui. No contexto de redes de comunicação, como a internet, é desejável que uma largura de banda maior seja fornecida, para permitir que um maior volume de dados seja transferido em um mesmo intervalo de tempo. A

unidade de medida usual é o mega bits por segundo (Mbit/s) e permite aos usuários de internet ter uma medida sobre o desempenho de cada um dos provedores de internet;

- Potência consumida: em uma época de preocupação com os recursos ambientais e sustentabilidade, além do próprio custo crescente da energia, a potência consumida pelos sistemas vem sido considerada uma importante métrica de desempenho. É comum observar que todos os equipamentos eletrônicos trazem consigo informações sobre seu consumo, possibilitando ao consumidor escolher qual o equipamento mais eficiente energeticamente, ou seja, com o menor consumo de eletricidade;

- Instruções por segundo: provavelmente uma das métricas mais tradicionais na avaliação de desempenho de uma arquitetura de processadores, fornecendo uma visão de quantos milhões (MIPS) ou bilhões (GIPS) de instruções são executadas por segundo em um processador. Existe muita controvérsia no uso desta medida (27), principalmente pela constante evolução das arquiteturas e de sua implementação interna, que mesmo suportando um mesmo conjunto de instruções, pode apresentar resultados bem diversos;

- Operações de ponto flutuante por segundo: este métrica é muito utilizada em sistemas que realizam operações científicas, como os supercomputadores para aplicações militares, previsão do tempo e outros eventos naturais. Com esta medição em unidade de FLOPS é possível estabelecer o posicionamento mundial de cada supercomputador, indicando qual é o desempenho de cada um deles e quanto de processamento máximo pode ser realizado;

- Ciclos de simulação: com o advento de modelos de alto nível para simulação de sistemas, passou-se a considerar o desempenho de cada uma destas implementações. Em sua medição qualquer métrica que possa ser simulada pode ser utilizada, como por exemplo, o número de ciclos de relógio simulados em um determinado tempo. Geralmente medido em uma escala de milhões de ciclos por segundo (MCPS), esta métrica fornece uma noção em termos do número de eventos que foram simulados, independente da tecnologia utilizada ou do sistema em questão,

podendo servir como uma forma genérica de avaliação dos diversos modelos de sistema de alto nível.

A combinação de uma ou mais métricas pode ser utilizada para avaliar o desempenho do sistema ou do modelo de alto nível, mas é importante que se tenha em mente quais aspectos possuem maior relevância na análise do sistema. Normalmente, a métrica escolhida deve refletir características importantes do sistema e desta forma permitir que uma análise comparativa seja realizada com outros sistemas que possuem funções similares. Com esta capacidade de comparação, é permitida uma avaliação de desempenho focada nos aspectos mais relevantes e com independência das diversas tecnologias utilizadas em implementações.

# 3 Estado da Arte

Para contextualizar este trabalho proposto, foi realizada uma ampla pesquisa do estado da arte, buscando trabalhos recentes e relevantes que confirmem a importância do que está sendo proposto, além de demonstrar que a proposta se dedica a resolver problemas ainda em aberto. Para melhorar a organização, o estado da arte foi dividido em três categorias: simulação nativa, abordagem híbrida e síntese de modelos, que serão dispostas nas seções a seguir. Por fim, uma análise comparativa irá sistematizar e retomar tudo que foi descrito, concentrando-se nos pontos chave, tornando mais simples e direto o entendimento geral de tudo que foi pesquisado.

## 3.1 Simulação Nativa

Esta é a área de maior relevância para este trabalho proposto, baseando-se fortemente no conceito de uso de modelos de sistema nativos (executam no sistema de desenvolvimento) para realização das simulações e análises de hardware e software na plataforma alvo. Diversos métodos vêm sendo desenvolvidos para aprimorar estes modelos, possibilitando comunicação rápida com TLM e interfaces para abstração de acesso ao hardware, sempre procurando obter estimativas precisas que normalmente só são obtidas com uso de ISS ou do próprio hardware real.

Em simulações puramente nativas, o software e o processador são modelos executados na máquina principal, utilizando as ferramentas de desenvolvimento disponíveis do próprio sistema. Como é ilustrado na figura 16, o modelo de software é executado pelo modelo do processador, suportando interface TLM para acesso ao barramento, além do suporte para o gerenciamento de interrupção, tanto em hardware como em software.

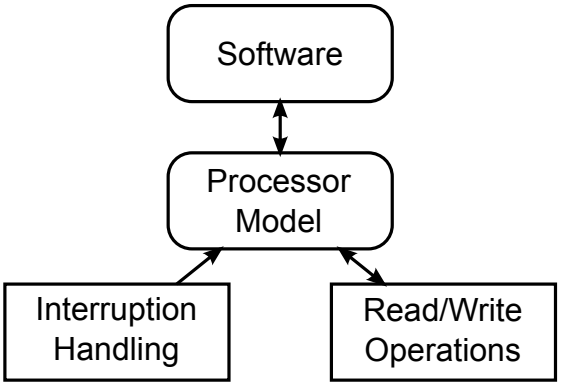


Figura 16: Fluxo de simulação nativa

### 3.1.1 Using a Dataflow Abstracted Virtual Prototype for HdS-design (Ecker et al. (1), IEEE 2009)

O software dependente de hardware (HdS) possui complexidade que cresce mais rápido do que a taxa de desenvolvimento do próprio circuito integrado que suporta, pois cada vez mais tarefas são migradas para o software, gerando uma pressão crescente para que novas metodologias de projeto tragam avanços em seu desenvolvimento. Os experimentos realizados obtiveram melhoria da ordem de 10 vezes no desempenho, no caso médio, quando comparando três modelos de simulação: o clássico ISS (ISS Virtual Platform ou ISS_VP), o emulador de instruções (Emulator Virtual Platform ou EMU_VP) e o modelo de abstração proposto para hardware e software (Hardware/Software Virtual Platform ou HWSW_VP).

#### 3.1.1.1 Fluxo de Desenvolvimento

A modelagem nível TLM é utilizada para desenvolvimento eficiente de uma plataforma de comunicação, onde são observadas as vantagens e desvantagens deste nível de modelagem, assim como um conjunto de requisitos que definem a abstração do fluxo de dados proposto por Ecker et al. (1).

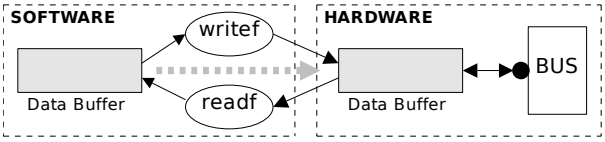


Figura 17: Fluxo de dados abstrato de Ecker et al. (1)

A abstração de fluxo de dados proposta, que pode ser vista na figura 17,

permite a abstração da interface de hardware/software e dos componentes de hardware através da transferência do vetor de dados (Data Buffer) diretamente do software para o modelo de hardware que irá fazer o acesso ao barramento e aos registradores. Neste modelo de hardware, o software é executado nativamente em um ambiente de emulação em SystemC, com dois modelos de tarefas (SC_THREAD): o primeiro que é responsável pela execução do software propriamente dito, invocando a função principal da aplicação; e outra que irá gerenciar as rotinas de interrupção de software ou Interruption Service Routines (ISR), com a suspensão do fluxo principal de execução. A interrupção do software é checada a cada acesso ao barramento, uma vez que não é possível interromper externamente a execução nativa das tarefas, ou seja, cada tarefa precisa permitir sua preempção explicitamente.

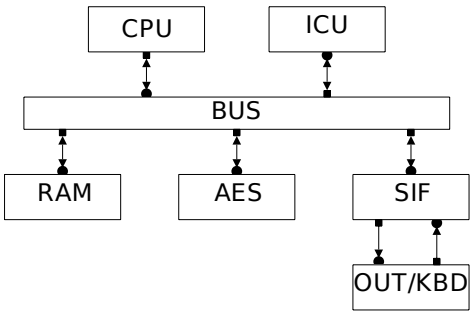


Figura 18: Plataforma virtual de Ecker et al. (1)

A plataforma virtual considerada, ilustrada na figura 18, é composta pelo unidade de processamento (CPU), gerenciador de interrupção (ICU), barramento de interconexão (BUS), unidade de memória (RAM), módulo de criptografia simétrica (AES) e uma unidade serial (SIF) que recebe comandos de uma teclado (KBD) e envia as saídas de texto para o terminal do usuário (OUT).

#### 3.1.1.2 Resultados

Na avaliação dos experimentos foi utilizado um servidor multiprocessado AMD Opteron de 64 bit, rodando Linux, utilizando a versão de SystemC 2.2 e a plataforma virtual foi compilada pelo GCC versão 4.1.2 com otimização de nível 2. Para fins de comparação, foram utilizados 3 modelos: o primeiro que é um ISS tradicional (ISS_VP), com precisão exata de tempo; o segundo com emulação do processador (EMU_VP) que executa o software nativamente, utilizando a ferramenta QEMU que traduz em tempo de execução as instruções

do modelo alvo; e a terceira opção que é a proposta (HWSW_VP) que é a abstração do fluxo de dados, onde o software acessa diretamente o modelo de hardware para realização de acesso aos dispositivos e gerenciamento de interrupção.

Tabela 1: Comparativo de simulação de Ecker et al. (1)

| Modelo | IO | AES | MIXED |
|---|---|---|---|
| ISS_VP | 1048,6s | 1632,7s | 1206,5s |
| EMU_VP | 49,9s | 84,9s | 48s |
| HWSW_VP | 0,2s | 32,5s | 3,6s |

Para os 3 modelos de simulação, listados na tabela 1, foram desenvolvidas também 3 aplicações: a primeira calcula todos os números primos até 1000 (IO) e armazena todos os números encontrados em um vetor de dados para serem codificados e decodificados pelo algoritmo AES, sendo transmitidos 1000 vezes pela interface serial (SIF) para exibição em terminal de texto; a segunda codifica e decodifica (AES) o vetor dos números primos, além de os imprimir no terminal de texto, por 1000 vezes; e a terceira aplicação (MIXED) os números primos são calculados 10 vezes e cada vez que o vetor é obtido, ele é encriptado e desencriptado por 10 vezes pelo AES, e por fim, o vetor resultante é impresso no terminal de texto por 10 vezes.

#### 3.1.1.3 Contextualização

O abordagem de Ecker et al. (1) tem como foco a construção de um modelo de alto nível para execução de sistemas que permita o desenvolvimento de HdS, em estágio inicial de desenvolvimento e operando com comunicação em nível de transações (TLM). Propondo uma interface para abstração do fluxo de dados, é possível construir HdS através de funções básicas de entrada e saída que irão realizar a transferência de dados entre o software e os dispositivos da plataforma. Com este recurso, o software não possui a capacidade de acesso direto aos registradores, ficando dependente de funções que precisarão ser desenvolvidas utilizando modelos ISS.

Por isto, considerando os objetivos propostos, a abordagem de Ecker et al. (1) não suporta o desenvolvimento completo de HdS, pois não permite a modelagem precisa de registradores para acesso aos dispositivos, somente fornecendo funções para entrada, saída e tratamento de interrupção. Outro

ponto que não é abordado é a precisão das simulações realizadas, visto que somente o desempenho é considerado, sem que se conheça o impacto da abstração nas informações e comportamentos obtidos.

### 3.1.2 Software Performance Simulation Strategies for High-level Embedded System Design (Wang et al. (2), Elsevier 2009)

A maioria das funções realizadas por um sistema embarcado são implementadas em software, por isto é importante que sejam geradas estimativas de desempenho destes sistemas no fluxo de projeto. Nas últimas décadas, os modelos baseados em ISS se tornaram uma parte essencial deste fluxo, apesar do baixo desempenho de simulação e alto detalhamento em sua construção. Com o advento de plataformas multiprocessadas e sua complexidade crescente, utilizar a estratégia ISS para exploração de espaço de projeto não é mais uma opção viável e muitas das novas abordagens procuram resolver estas limitações com execução nativa.

#### 3.1.2.1 Fluxo de Desenvolvimento

O conceito do fluxo proposto é a geração, para cada programa, de um modelo de simulação que execute nativamente na máquina principal com alto desempenho, mas produzindo informações de execução referentes a plataforma alvo. Surgem 3 problemas nesta estratégia: representação funcional, sendo realizada com o próprio código fonte da implementação do software; a análise de tempo, feita pelo processamento de informações do código binário alvo obtido; e o acoplamento da representação funcional e do modelo de desempenho, gerando os atrasos de tempo simulado em relação ao tempo de execução nativo.

Como pode ser visto na figura 19, o fluxo de desenvolvimento é composto por 3 macro etapas: geração de código fonte intermediário ou Intermediate Source Code (ISC) Generation, que realiza a compilação para plataforma alvo (cross-comp - frontend) e preparação do código (IR to ISC conversion), com estruturas de depuração para obtenção de informações de tempo; instrumentação (Instrumentation), que recebe o código fonte preparado para ser novamente compilado, sendo que desta vez será utilizando em um modelo

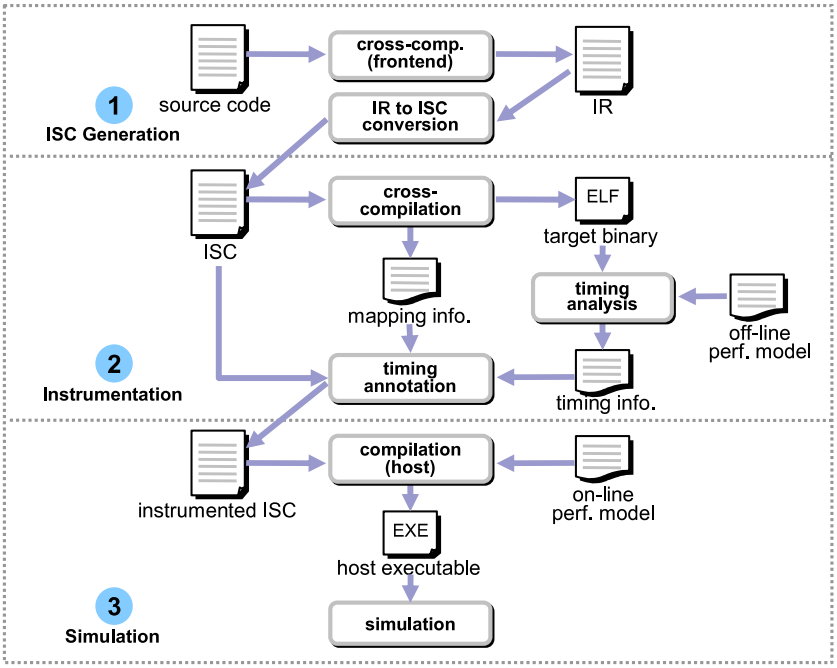


Figura 19: Fluxo de desenvolvimento de Wang et al. (2)

de desempenho estático (off-line perf. model), para extração de informações de tempo e realização de instrumentação do tempo no código fonte intermediário; e simulação (Simulation) que compila nativamente o código fonte intermediário instrumentado (instrumented ISC), utilizando o modelo dinâmico de desempenho, para ser executado e gerar as estimativas de tempo simulado necessárias.

#### 3.1.2.2 Resultados

Na preparação dos experimentos, foram feitas simulações para se verificar o impacto da geração do código fonte intermediário na precisão de tempo e foi constatado que, comparado ao código fonte original, utilizando ISS, um erro máximo de 1,92% foi obtido, com média de erro absoluta de 0,63%. Com estes dados, fica constatado que o código intermediário altera as propriedades de tempo da aplicação original, mas que estas mudanças são muito pequenas e pode ser desconsideradas.

Foram utilizados 6 aplicações e 4 abordagens de simulação nos experimentos (como pode ser visto na tabela 2), analisando principalmente o erro obtido pelas estimativas e utilizando o ISS como referência, mesmo sabendo que existe um pequeno erro associado ao código intermediário gerado. As apli-

Tabela 2: Comparativo de simulação de Wang et al. (2)

| Referência | Abordagem de Simulação | | | | | |
|---|---|---|---|---|---|---|
| | BLS | | SLS | | iSciSim | |
| | Estimativa | Erro | Estimativa | Erro | Estimativa | Erro |
| fibcall | 4384 | 0% | 12973 | 196% | 4356 | 0,64% |
| insertsort | 630 | 0% | 975 | 55% | 631 | 0,16% |
| bsearch | 59010 | 0% | 75013 | 27% | 58010 | 1,69% |
| crc | 17201 | 0% | 19385 | 12,7% | 17313 | 0,65% |
| blowfish | 262434 | 0% | 265735 | 1,26% | 260899 | 0,58% |
| aes | 3624426960 | 0% | 3624685359 | 0,01% | 3544422940 | 2,21% |

cações utilizadas são referências conhecidas (fibcall, insertsort, bsearch, crc, blowfish e aes) e os níveis de simulação são: ISS, Binary Level Simulation (BLS), Source Level Simulation (SLS) e a abordagem proposta (intermediate Source code instrumentation based Simulation ou iSciSim).

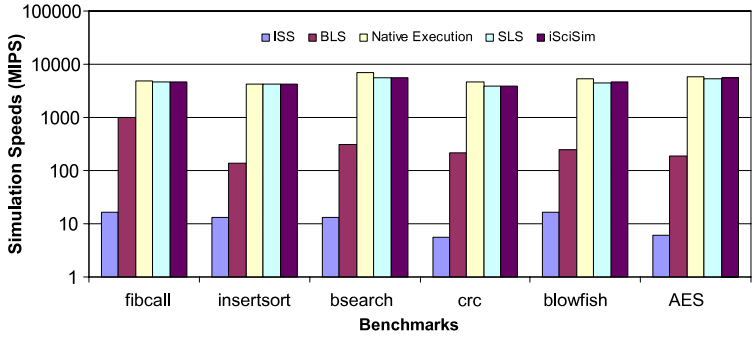


Figura 20: Velocidades de simulação de Wang et al. (2)

Os números de instruções estimadas, juntamente com o tempo de execução total, é capaz de nos fornecer a taxa de milhões de instruções executadas por segundo ou Million Instructions Per Second (MIPS), que é uma unidade absoluta de medida de desempenho em uma determinada arquitetura. Na figura 20, todos os 4 modelos de simulação são considerados, rodando cada uma das 6 aplicações de referência escolhidas.

A abordagem proposta iSciSim atinge desempenho equivalente a execução nativa, mas mantendo um nível de erro próximo ao ISS, fornecendo uma solução de baixa complexidade de entendimento e utilização. Além disto é rápida o suficiente para permitir uma exploração de espaço de projeto eficiente e precisa o bastante para garantir a medição da influência das escolhas de projeto em um alto nível de abstração.

#### 3.1.2.3 Contextualização

O trabalho de Wang et al. (2) ressalta a importância da análise antecipada do comportamento do software, principalmente por sua crescente complexidade e importância em plataformas modernas. Também é colocado que é inviável continuar utilizando modelos ISS para desenvolvimento, por seu custo de construção e por seu baixo desempenho de simulação. Para resolver estes problemas, foi proposto um modelo nativo de simulação que é instrumentado com informações de tempo da arquitetura alvo, para permitir estimativas precisa sobre a execução.

Dentre os objetivos traçados, a execução nativa do código fonte da aplicação está em total sincronia com o que está sendo proposto por Wang et al. (2), permitindo que a implementação de software seja avaliada com alto desempenho e precisão. Por outro lado, apesar do suporte previsto ao HdS é necessário que código adicional para ser desenvolvido para seu funcionamento, sem nenhum tipo de facilitador oferecido pela abordagem, estando definidos como fora do escopo do trabalho.

### 3.1.3 Cycle-Count-Accurate Processor Modeling for Fast and Accurate System-Level Simulation (Lo et al. (3), EDAA 2011)

Os ambientes tradicionais de desenvolvimento, que utilizam modelos com precisão de ciclo ou Cycle-Accurate (CA) ou com aproximação de ciclo ou Cycle-Approximate (CX), recaem em baixo desempenho de simulação, pelo excessivo detalhamento das informações de tempo, ou em baixa precisão, por simplificar demasiadamente o modelo de aproximação, respectivamente. O trabalho de Lo et al. (3) busca uma modelagem que ofereça alto desempenho de simulação, mas mantendo a precisão de tempo exata para simulações de sistema, através de um modelo de processador de contagem de precisão de ciclos ou Cycle-Count-Accurate (CCA) que abstrai os modelos internos de pipeline e cache em modelos com contagem de ciclos de tempo.

### 3.1.3.1 Fluxo de Desenvolvimento

Na abordagem de contagem de ciclo aproximada ou CCA, é atingido o caminho do meio entre as ideias de precisão de ciclo e de aproximação de ciclo, melhorando o desempenho da simulação e reduzindo os níveis de detalhe de tempo a um nível que não comprometam a precisão esperada. No fluxo de desenvolvimento proposto, o modelo CCA atende ao requisito que todas as operações de acesso as interfaces, como barramento, serão corretas tanto no funcionamento como na precisão de tempo, abstraindo completamente os detalhes internos.

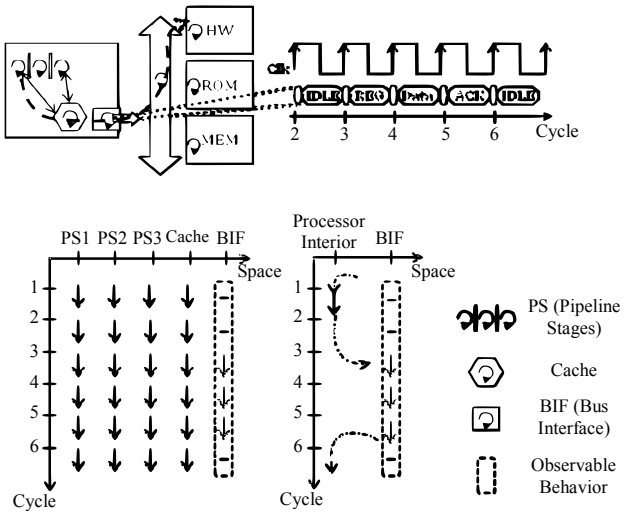


Figura 21: Modelo abstrato de Lo et al. (3)

Na figura 21, este modelo abstrato de processador é melhor detalhado, onde a precisão é implementada considerando o princípio da observabilidade, ou seja, do que será observado pelo projetista do sistema (neste caso as interfaces) e todo os demais detalhes serão eliminados para melhorar o desempenho. Neste exemplo fornecido na figura 21, o processador acessa a um componente de hardware da plataforma, utilizando a interface com o barramento (BIF) que é o ponto de observação do modelo para verificação de precisão e de tempo.

Os estágios de pipeline (PS) e a cache, que representam o comportamento interno e não observável do sistema, são convertidos para um modelo CCA (Processor Interior) que abstrai todos os detalhes, fornecendo uma contagem de ciclos equivalente para que o comportamento observado em BIF seja o mesmo. Esta abstração é realizada pelo subsistema de modelagem de

pipeline ou Pipeline Subsystem Modeling (PSM) e pelo subsistema de modelagem de cache ou Cache Subsystem Model (CSM).

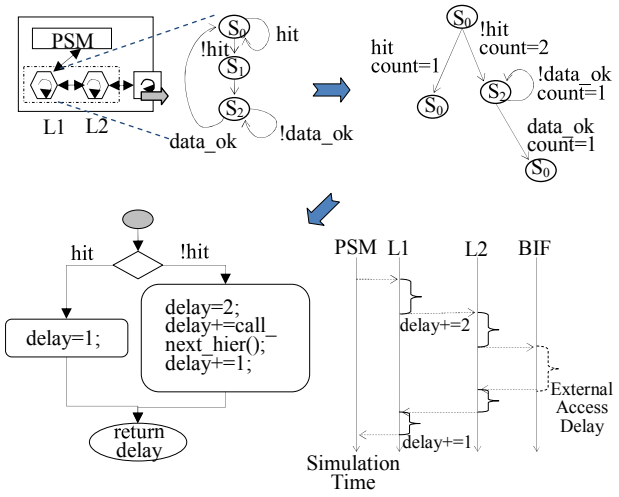


Figura 22: Máquina de estado de cache de Lo et al. (3)

No PSM, são construídos modelos de comportamento de blocos básicos do programa a partir do grafo de controle de fluxo ou Control Flow Graph (CFG), para a realização da análise de comportamento e tempo estáticos. Na etapa de análise dinâmica, cada um dos blocos básicos tem seu comportamento e tempo de contagem de ciclos, obtidos estaticamente, simulados para execução em ordem do programa equivalente. Com o comportamento interno modelado, é preciso fazer o mesmo para a cache através do CSM, implementando com precisão o comportamento das operações de acerto (hit) ou de falta (miss). No modelo de cache é utilizada uma máquina de estados finito com relógio ou Clocked Finite State Machine (CFSM), como é ilustrado na figura 22, que implementa o comportamento da cache e implementa os atrasos necessários para cada operação realizada.

#### 3.1.3.2 Resultados

Como estudo de caso foi escolhido o processador OpenRISC 1200 (OR1200), devido a sua natureza de código aberto com todos os detalhes de projeto disponíveis, o que torna a verificação dos modelos gerados relativamente fácil. Este processador utiliza a arquitetura Harvard, com 5 estágios de pipeline e paradigma RISC com 32 bits, tendo desempenho comparável ao seu competidor ARM9.

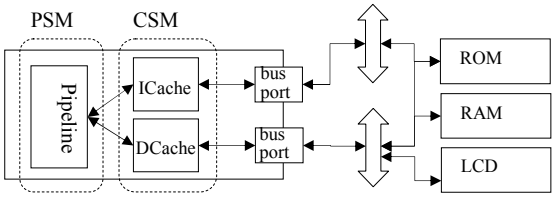


Figura 23: Plataforma OR1200 de Lo et al. (3)

A maioria das aplicações de referência utilizadas era do próprio OR1200, mas foi adicionado um decodificador de 32 quadros MPEG-4 QCIF que realiza a busca da imagem de uma ROM e a exibe em uma tela de LCD, como pode ser visto na arquitetura da plataforma na figura 23.

Tabela 3: Comparativo de simulação de Lo et al. (3)

| Casos de Teste | Tempo de Análise | CA Tradicional | CA Compilado | CCA Proposto | Melhoria Desempenho |
|---|---|---|---|---|---|
| fib | 0,12 | 1,33 | 2,69 | 68,31 | 51x |
| mul | 0,13 | 1,42 | 2,67 | 53,72 | 38x |
| cbasic | 0,25 | 1,55 | 2,66 | 62,56 | 40x |
| dhry | 0,33 | 1,94 | 2,88 | 117,12 | 60x |
| mpeg4 | 2,60 | 1,93 | 2,87 | 114,51 | 59x |

Na tabela 3, as velocidades de simulação são expressas em Milhões de Ciclos por Segundo (MCPS), comparando-se o modelo tradicional CA, o modelo CA compilado e o modelo CCA proposto. O tempo de análise obtido é medido em segundos e é referente ao tempo gasto para análise e geração do modelo CCA, crescendo linearmente com o aumento do tamanho do número de blocos básicos da aplicação em questão. A melhoria de desempenho obtida gira em torno de 50 vezes, quando comparado ao modelo CA tradicional, significando em termos práticos a redução de uma simulação que dura mais de 6 dias para apenas 3 horas.

#### 3.1.3.3 Contextualização

O trabalho de Lo et al. (3) confronta alguns dos problemas mais relevantes desenvolvimento de software: o baixo desempenho das simulações com ISS, abstrair modelos de execução com precisão e a cultura de desenvolvimento baseada em código, ao invés de modelos de sistema. Como solução foi proposta uma abordagem de CCA que permite uma abstração exata, sob uma perspectiva de caixa preta do sistema, do ISS clássico, sem causar impacto

na precisão gerada e sem demandar do projetista novas habilidades para o projeto de sistemas.

A proposta de Lo et al. (3) está fortemente alinhada com os objetivos propostos, possuindo todos os requisitos que foram enumerados, basicamente por apresentar uma representação exata e mais rápida do que um modelo baseado em ISS. Porém, os resultados obtidos, apesar de positivos e mostrarem ganhos de cerca de 50 vezes sobre simulações com ISS, ainda estão muito abaixo do desempenho que pode ser alcançado por plataformas virtuais que utilizam modelos nativos.

### 3.1.4 Automatic Timing Granularity Adjustment for Host-Compiled Software Simulation (Razaghi et al. (4), IEEE 2012)

Nas abordagens de simulação nativa (host-compiled) mais recentes os modelos têm sido desenvolvidos com mecanismos precisos de tempo de execução. A estratégia utilizada consiste em instrumentar o código que será executado nativamente com informações de tempo que irão determinar os atrasos de cada trecho do código. No trabalho de Razaghi et al. (4) é proposto a eliminação do compromisso entre desempenho e precisão de tempo estimado, utilizando execução de software nativa. Para tanto, é proposto um mecanismo de ajuste automático e dinâmico de granularidade de tempo que permite a eliminação dos erros de temporização e mantém o alto desempenho da simulação, de aproximadamente 900 MIPS.

#### 3.1.4.1 Fluxo de Desenvolvimento

A estratégia proposta consiste na utilização de simulador nativo de software em alto nível de abstração, organizado em camadas bem definidas, como está ilustrado na figura 24. Nesta arquitetura, estão dispostas 5 camadas para execução nativa do sistema:

- Aplicação (Application): consiste de processos concorrentes e sequenciais descritos em uma SLDL, se comunicando uns com os outros através de canais abstratos da SLDL. Para integração desta aplicação com o sistema é utilizada uma interface genérica de SO provida pelo modelo de

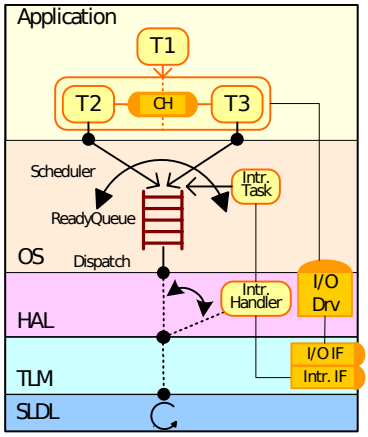


Figura 24: Arquitetura do simulador nativo de Razaghi et al. (4)

sistema operacional;

- Sistema Operacional (OS): faz a replicação de uma típica estrutura de SO, como forma de permitir e gerenciar a execução de uma aplicação de múltiplas tarefas. Este modelo escalona, enfileira, despacha e executa a aplicação e suas tarefas de interrupção, seguindo as políticas de escalonamento definidas;

- Camada de Abstração de Hardware (HAL): esta camada inclui todos os gerenciadores de entrada e saída de dispositivos, além de implementar um mecanismo abstrato de gerenciamento de interrupção. Quando uma interrupção é capturada pela camada TLM, a camada HAL suspende a execução da aplicação e permite que a tarefa de tratamento seja iniciada pelo SO;

- Comunicação em Nível de Transação (TLM): permite que seja realizada a comunicação, através de transações, entre o sistema e a plataforma virtual, definindo as interfaces pelas quais podem ser realizadas entradas e saída, além das requisições de interrupção da plataforma;

- Linguagem de Descrição em Nível de Sistema (SLDL): funciona como infraestrutura básica, tanto para o sistema como para a plataforma, permitindo que as simulações sejam executadas com noção de tempo simulado e diversos construtores possam ser aplicados em todas as camadas superiores.

Com as técnicas propostas para controle de tempo é possibilitado que o núcleo do SO permita que uma tarefa execute e faça o acumulo de atrasos,

sem chamar o escalonador ou avançar o tempo de simulação. Este recurso permite que mesmo em alta granularidade de tempo, o desempenho não seja prejudicado. Isto ocorre basicamente pelo fato de cada tarefa ter seu próprio controle do tempo que precisa ser simulado e este tempo só é de fato contabilizado quando a preempção da tarefa é atingida.

#### 3.1.4.2 Resultados

Para obtenção dos resultados experimentais, foi aplicada a SLDL SpecC e o modelo de RTOS em uma aplicação telefonia móvel de porte industrial. Esta plataforma é baseada na arquitetura ARM7 e o sistema em questão desempenha funções concorrentes de decodificação de MP3, de codificação de JPEG e de tarefas de controle.

Tabela 4: Comparativo de simulação de RTOS de Razaghi et al. (4)

| Métrica | RTOS 1 us | RTOS 10 us | RTOS 100 us | RTOS 1000 us |
|---|---|---|---|---|
| Erro (MP3) | 0,73% | 0,79% | 1,40% | 9,65% |
| Erro (JPEG) | 7,33% | 7,33% | 7,33% | 7,35% |
| Erro (MP3 + JPEG) | 4,18% | 4,20% | 4,49% | 8,45% |
| Desempenho | 340 MIPS | 790 MIPS | 930 MIPS | 1080 MIPS |
| Tempo de Execução | 0,61s | 0,26s | 0,22s | 0,19s |

Na tabela 4 são exibidos os resultados de erro médio, desempenho e tempo de execução considerando um modelo de sistema de RTOS, sem as técnicas propostas. São definidos 4 níveis de granularidade, variando de 1 até 1000 us, e conforme esperado, quanto maior a granularidade de tempo utilizada, maior o desempenho obtido. Entretanto, conforme também previsões, a medida que são utilizados passos maiores de tempo, o erro tende a ser majorado.

Tabela 5: Comparativo de simulação de ATGA de Razaghi et al. (4)

| Métrica | ATGA 1 us | ATGA/Acc 1 us | ATGA Evento | ATGA/Acc Evento |
|---|---|---|---|---|
| Erro (MP3) | 0,73% | 0,74% | 0,73% | 0,74% |
| Erro (JPEG) | 7,33% | 7,32% | 7,33% | 7,32% |
| Erro (MP3 + JPEG) | 4,18% | 4,18% | 4,18% | 4,18% |
| Desempenho | 554 MIPS | 684 MIPS | 621 MIPS | 892 MIPS |
| Tempo de Execução | 0,37s | 0,30s | 0,33s | 0,23s |

Considerando os mecanismos propostos para controlar a granularidade

automaticamente (ATGA), também são feitos 4 tipos de experimentos com granularidade de 1 us e baseado em evento, além de considerar a técnica de acumulação de tempo (Acc), conforme por ser visto na tabela 5. É importante perceber que com a mesma granularidade mínima de 1 us e baseando em técnicas de acumulação de tempo ou por eventos, houve uma estabilização do erro médio em todos os casos.

#### 3.1.4.3 Contextualização

O trabalho de Razaghi el al está dedicado a apresentar um conjunto de técnicas para controle automático de granularidade de tempo (ATGA), com o objetivo principal de melhorar o desempenho obtido e manter uma alta resolução de granularidade. Tudo isto aplicado em um contexto de simulação nativa (host-compiled), proporcionado pelo uso de SLDL para construção dos modelos e inclusive do próprio software.

Contextualizando o trabalho de Razaghi et al. (4) com os objetivos propostos, é notório como é relevante a necessidade de utilizar modelos de alto nível para atender as demandas de sistemas cada vez mais complexos. Porém, nos objetivos traçados se procura definir um modelo que abstraia a execução da implementação de software, sem utilização de modelos nem de ferramentas de síntese. Este aspecto é essencial para simulação de software em nível de sistema, com suporte para desenvolvimento de HdS, sem ter que obrigatoriamente recorrer para complexos e lentos modelos ISS.

### 3.1.5 Abstract System-Level Models for Early Performance and Power Exploration (Gerstlauer et al. (5), IEEE 2012)

Os modelos de alto nível de sistema proporcionam um rápido retorno aos projetistas sobre o efeito de suas decisões, permitindo a avaliação de métricas críticas do sistema como: desempenho, consumo de potência e custo do sistema. Devido ao seu comportamento inerentemente dinâmico e suas complexas iterações, é bastante difícil de serem realizadas estas análises de forma estática, assim tornando indispensável o uso de modelos executáveis.

Existem diversas abordagens para realização de modelagem de sistemas, podendo estar organizadas de acordo com a granularidade da computação

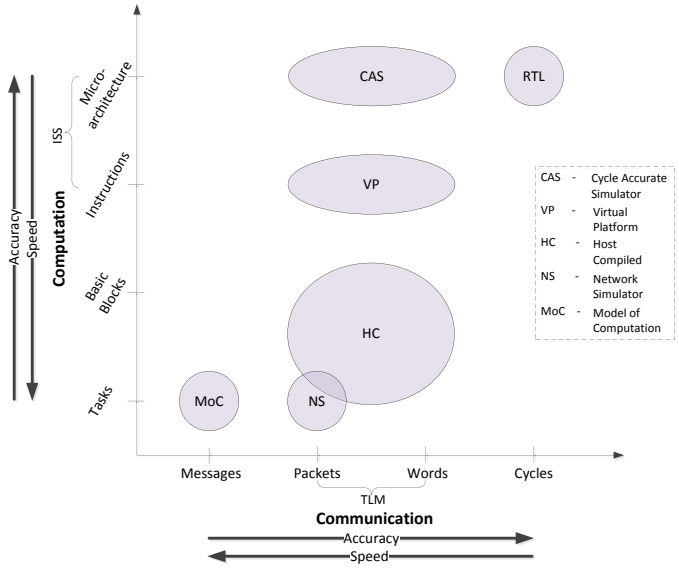


Figura 25: Espaço de modelagem de Gerstlauer et al. (5)

e da comunicação, bem ilustradas na figura 25. Dentre estes vários níveis possíveis de granularidade, será focada a atenção no domínio de simulação nativa (Host Compiled ou HC) que realiza suas computações em nível de bloco básico e a comunicação ocorre em nível de transações (TLM).

#### 3.1.5.1 Fluxo de Desenvolvimento

A proposta de fluxo de desenvolvimento, que pode ser visualizada na figura 26, com simulação nativa, tem início com a escolha de um código fonte C (C Source Code) que será compilado, usando uma ferramenta genérica de compilação, no caso o GCC. Após realização das otimizações, é feita a geração de uma representação intermediária (IR) que permite que as ações de otimização tomadas sejam conhecidas e as estimativas e anotações possam ser realizadas.

Através da conversão desta representação intermediária para código C (IR to C), o código novamente passa a ser compilável e já contém as informações anotadas em sua estrutura. Existe uma suposição de que uma vez novamente compilado (Compiler Backend) o código não sofrerá alterações significativas no grafo de controle fluxo (CFG) da aplicação. O próximo passo está em relacionar os endereços gerados para o código binário (Binary) e os blocos básicos anotados na representação intermediária, obtendo-se no final um modelo executável nativo do sistema (Host-Compiled Model).

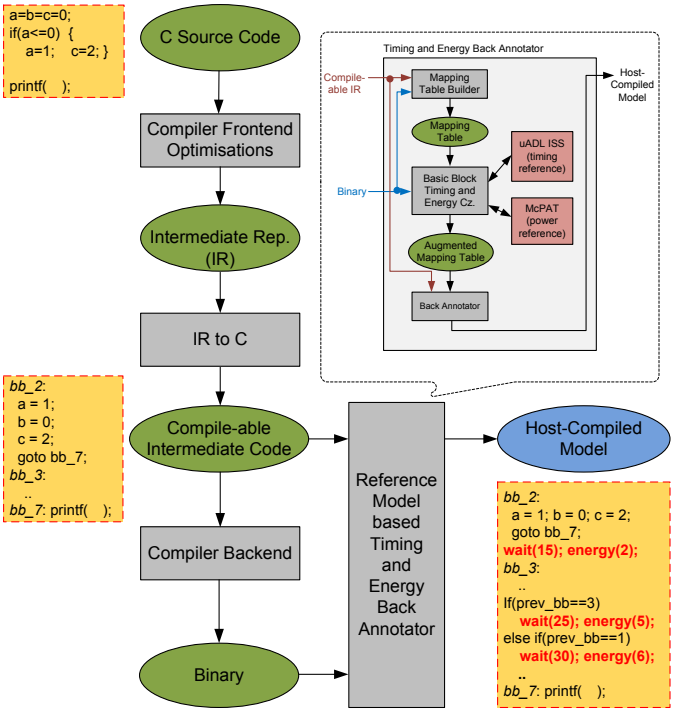


Figura 26: Fluxo de anotação de código de Gerstlauer et al. (5)

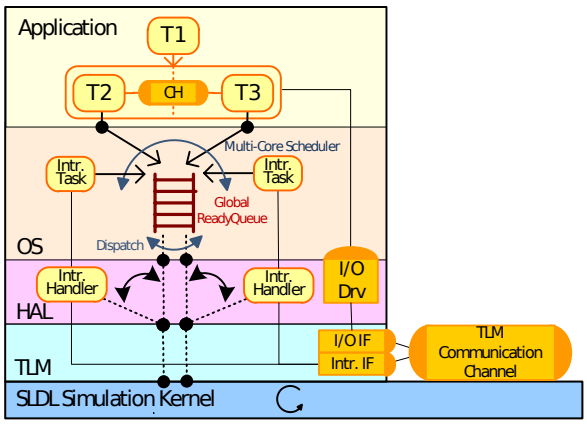


Figura 27: Modelo de plataforma de Gerstlauer et al. (5)

Para suportar aplicações com múltiplas tarefas, é utilizado um modelo abstrato de RTOS que emula a execução de tarefas de alto nível, ou seja, utilizando um modelo de aplicação em SLDL. Estes serviços são fornecidos por uma interface genérica do modelo de OS que permite a inicialização, gerenciamento das tarefas, modelagem de atraso de tempo e sincronização de eventos. Sua implementação interna é feita para utilizar os núcleos disponíveis no sistema nativo e possui uma série de parâmetros para permitir uma avaliação das possíveis configurações disponíveis do sistema.

Conforme visto na figura 27, juntamente a este modelo de OS, estão os modelos da aplicação (Application), da camada de abstração de hardware

(HAL) e da comunicação em nível de transação (TLM), todas sendo suportadas por um núcleo de simulação da SLDL.

#### 3.1.5.2 Resultados

Na realização de experimentos para análise da estratégia de anotação de informações no código, foi feita uma comparação do tempo e potências estimadas com relação aos resultados obtidos no ISS e na referência de potência (McPAT). Neste primeiro momento foi utilizada uma aplicação personalizada chamada "Eratosthenes' Sieve", para validar o modelo proposto, com o funcionamento dedicado encontrar número primos de uma faixa de 0 até 500000.

Tabela 6: Comparativo de simulação de Gerstlauer et al. (5)

| Métrica | Simulação Nativa | ISS/McPAT | Erro |
|---|---|---|---|
| Tempo em ciclos | 22.864.740 | 22.864.730 | 0,0004% |
| Potência em mJ | 222,6 | 241,4 | 7,8% |
| Desempenho em MIPS | 2000 | 0,8 | - |

Na tabela 6, os resultados de tempo e de potência são exibidos, além das informações de desempenho e de erro gerados. Focando no aspecto temporal, foi percebido um pequeno erro de apenas 10 ciclos na estimativa gerada, o que representa um erro bem pequeno de 0,0004%. Já na potência estimada, o erro foi bem mais significativo, com cerca de 8% de desvio, quando comparado ao modelo ISS/McPAT. Trabalhando agora em nível mais elevado de abstração, a aplicação agora consiste de tarefas de alto nível que interagem com o modelo de SO para escalonamento de suas tarefas entre os núcleos de processamento disponíveis. Para avaliar o desempenho e precisão neste contexto foi utilizada como referência uma plataforma virtual com dois processadores MIPS34Kc Malta, executando Linux 2.6.24 SMP.

Nos experimentos, um conjunto aleatório de tarefas periódicas, com períodos entre 1 e 100 ms, foram geradas, com diferentes tamanhos (pequeno, médio e grande) para ajustar a carga de utilização de cada processador. Na tabela 7, estes conjuntos de tarefas de diferentes tamanhos são executados, permitindo que seja analisadas o número médio de tarefas por núcleo, a média de utilização de cada núcleo, os erros médios obtidos e desempenhos com diferentes níveis de granularidade de tempo do RTOS (1, 10, 100 e 1000

Tabela 7: Comparativo de simulação de RTOS de Gerstlauer et al. (5)

| Métrica | Pequeno | Médio | Grande |
|---|---|---|---|
| Tarefas por núcleo | 11 | 4 | 3 |
| Utilização do núcleo | 0,6 | 0,7 | 0,7 |
| Erro (1 us) | 0,5% | 0,25% | 0,11% |
| Erro (10 us) | 0,71% | 0,22% | 0,10% |
| Erro (100 us) | 0,64% | 0,69% | 0,43% |
| Erro (1000 us) | 10,3% | 6,46% | 4,0% |
| Desempenho (1 us) | 0,15 GIPS | 0,13 GIPS | 0,13 GIPS |
| Desempenho (10 us) | 1,68 GIPS | 1,3 GIPS | 1,17 GIPS |
| Desempenho (100 us) | 10,5 GIPS | 8,2 GIPS | 8 GIPS |
| Desempenho (1000 us) | 23 GIPS | 31 GIPS | 36 GIPS |

microsegundos).

#### 3.1.5.3 Contextualização

Contextualizando o trabalho de Gerstlauer et al. (5) nos objetivos propostos, é possível ver como é relevante o desenvolvimento de modelos que permitam a execução de software embarcado nativamente, melhorando drasticamente seu desempenho, mas mantendo baixos níveis de erro nas estimativas geradas. Outro ponto interessante é o uso de aplicação em nível de código fonte para avaliação de desempenho e de potência, através de anotação de código em nível de bloco básico, permitindo que a própria implementação do software seja avaliada.

Quando é inserido o ambiente de múltiplas tarefas, o software é organizado em um conjunto de tarefas de alto nível suportados por uma SLDL, utilizando um modelo de RTOS genérico para gerenciamento das operações. Neste ponto, a proposta de Gerstlauer et al. (5) se diferencia dos objetivos listados, pois utiliza a própria implementação em código fonte do software e dar um suporte ao desenvolvimento de HdS, ao invés de utilizar um modelo de HAL. Nos objetivos propostos está previsto a simulação nativa do software da aplicação e da HAL em um modelo de processador de alto nível, permitindo que a implementação seja avaliada diretamente, sem necessidade de utilização de modelos ISS.

### 3.1.6 On the interfacing between QEMU and SystemC for virtual platform construction: Using DMA as a case (Yeh et al. (6), Elsevier 2012)

A co-simulação de componentes de hardware e de software em um ambiente de plataforma virtual demanda a utilização de interfaces para o acesso de memória, para realização de operações de E/S e para o gerenciamento de interrupções. Para suportar estes requisitos, o trabalho de Yeh et al. (6) utiliza modelos de hardware modelados na linguagem SystemC e um simulador em nível de instruções baseado no QEMU (56).

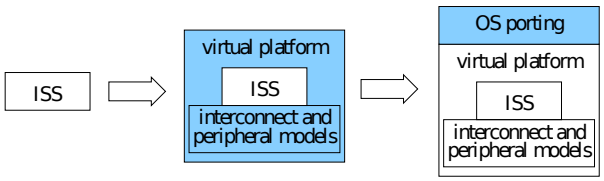


Figura 28: Plataforma convencional baseada em ISS

Em uma abordagem baseada em ISS integrado no ambiente de plataforma virtual, ilustrada na figura 28, todos as funcionalidades da plataforma precisam ser modelados em uma SLDL para permitir que o código binário do sistema possa ser executado sem modificações ou porte do SO.

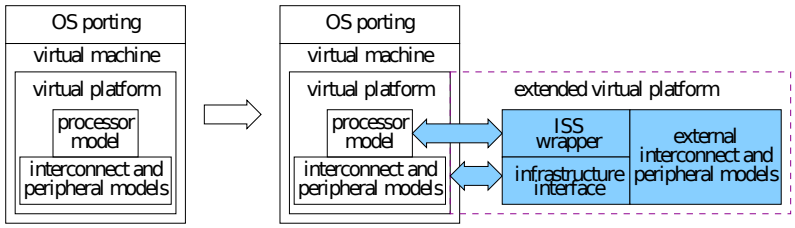


Figura 29: Plataforma baseada em máquina virtual

Ao invés de modelar todos os aspectos do sistema em um ISS, a abordagem de máquina virtual utilizando o QEMU realiza a tradução dinâmica do código binário da arquitetura alvo em tempo de execução para o sistema nativo de desenvolvimento. Esta tradução ocorre em nível de blocos básicos que são mapeados em uma cache de 16 MB e utilizam uma alocação fixa de registradores, além de realizar o gerenciamento de memória e o tratamento de interrupções.

### 3.1.6.1 Fluxo de Desenvolvimento

A utilização da abordagem proposta por Yeh et al. (6) tem como objetivo a utilização da infraestrutura do QEMU para execução do código binário do sistema sem modificações e a geração de uma interface entre o ambiente nativo do QEMU e o ambiente virtual de simulação baseado em SystemC. Para implementar a comunicação entre estes dois ambientes, é utilizada a comunicação baseada em fila unidirecional, como pode ser visto na figura 30, além de portas para acesso de instruções e de dados e de módulos para interface com a memória e para interrupção.

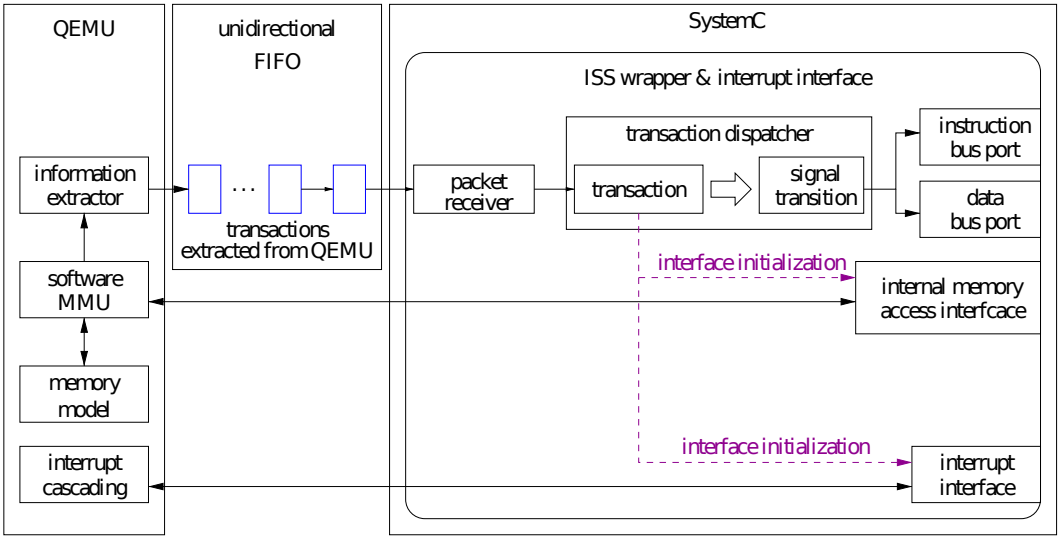


Figura 30: QEMU integrado em um ambiente SystemC

As principais contribuições desta organização, com relação aos demais trabalhos baseados em QEMU, está na possibilidade de utilização de dispositivos mestres na plataforma, como o Acesso Direto à Memória (DMA), que facilitam o co-projeto dos modelos de hardware e dos gerenciadores de dispositivos em um estágio precoce de desenvolvimento. Outro aspecto que está sendo explorado no trabalho é a avaliação do desempenho da arquitetura do sistema, utilizando uma versão com e sem suporte de DMA para transferência de dados.

### 3.1.6.2 Resultados

Nos experimentos realizados foi modelado em SystemC um módulo de DMA PrimeCell PL080 que, por não possuir gerenciador de dispositivo para Linux, pre-

cisou ser desenvolvido para realização das simulações. O ambiente virtual de simulação utiliza a plataforma Versatile/PB926EJ-S do QEMU, que é baseada na família ARM9 de processadores para execução do sistema. O desempenho foi avaliado de duas formas distintas: medindo o tempo de inicialização do sistema operacional Linux e analisando as informações coletadas durante o processo de inicialização.

Para obtenção dos resultados foram realizadas 30 simulações de inicialização do Linux com e sem utilização de transferência de dados por DMA. Foi observado que para cada palavra transferida pelo DMA foi evitada a execução de 9,05 até 12,36 instruções pelo processador, o que impediu a perda de desempenho do sistema com o DMA habilitado.

#### 3.1.6.3 Contextualização

A proposta de Yeh et al. (6) procurou suportar o desenvolvimento integrado de hardware e de software através de uma abordagem de simulação semi-nativa que consiste em utilizar as instruções nativas do sistema de desenvolvimento para implementar as instruções de uma determinada arquitetura alvo, utilizando a tradução dinâmica de código em nível de blocos básicos.

O alto nível de detalhe fornecido pela emulação do QEMU permite avaliar com precisão de instrução o comportamento do sistema, gerando uma melhoria para o estado da arte ao evitar a complexa modelagem do ISS e ao verificar precocemente o comportamento do sistema, mesmo sem ter a plataforma de hardware disponível. Entretanto, o desempenho de simulação obtido é bastante inferior com relação a outras abordagens baseadas em interpretação de instruções e em simulação compilada, basicamente pela grande quantidade de informações geradas decorrentes dos acessos à memória e das operações de E/S realizadas.

## 3.2 Modelos Híbridos de Simulação

Quando se fala em uma abordagem híbrida, é natural pensar em uma estratégia que combina o melhor de duas ou mais soluções. Este caso não é diferente, pois os trabalhos pesquisados particionam o sistema em uma parte

independente e em outra dependente de plataforma, buscando combinar suas melhores características. Se valendo do uso de modelos rápidos para execução da porção independente de plataforma, são obtidos ganhos de desempenho da ordem de 100 vezes, quando comparadas a estratégica que utiliza unicamente modelos baseados em ISS. A parte do sistema dependente de plataforma continua sendo simulada em um ISS e se comunica com o modelo rápido, de forma a manter uma baixa taxa de erro das estimativas.

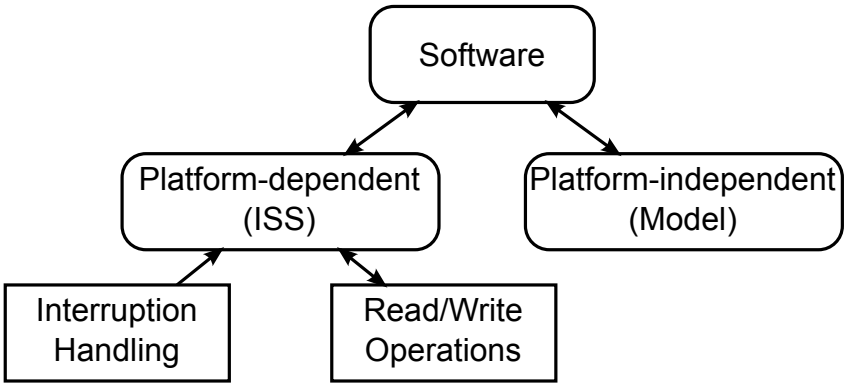


Figura 31: Fluxo de uso de modelos híbridos

O fluxo de desenvolvimento é ilustrado na figura 31, onde o software é processado por um conjunto de ferramentas e particionado em duas unidades: uma dependente de plataforma, contendo o software assembly e acesso ao hardware; e outra independente de plataforma que é responsável pelas funcionalidades do sistema, como aplicações e tarefas de controle e escalonamento de tarefas do sistema operacional.

### 3.2.1 HySim: A Fast Simulation Framework for Embedded Software Development (Kraemer et al. (7), ACM 2007)

Neste trabalho é proposto um modelo de simulação híbrida, o HySim que permite a combinação de execução nativa com ISS do software embarcado. Isto é obtido com o particionamento do software em duas unidades: uma que é chamada de software independente de plataforma que será executado nativamente no computador principal; e outra que é denominada de software dependente de plataforma que será executada no ISS. Os principais objetivos do trabalho de Kraemer et al. (7) são: desempenho de simulação, estimativas de desempenho por aproximação de tempo simulado, compatibilidade binária e não interferência na execução do ISS.

#### 3.2.1.1 Fluxo de Desenvolvimento

Conhecendo cada um dos objetivos do trabalho de Kraemer et al. (7), o fluxo de desenvolvimento (que pode ser visualizado na figura 32) tem início com o código fonte em C, para atender aos objetivos de compatibilidade e transparência. O código de entrada é dividido na parte dependente de plataforma, no ramo esquerdo (Target Compiler) e na parte independente de plataforma, no ramo direito (Instrumenter & Host Compiler).

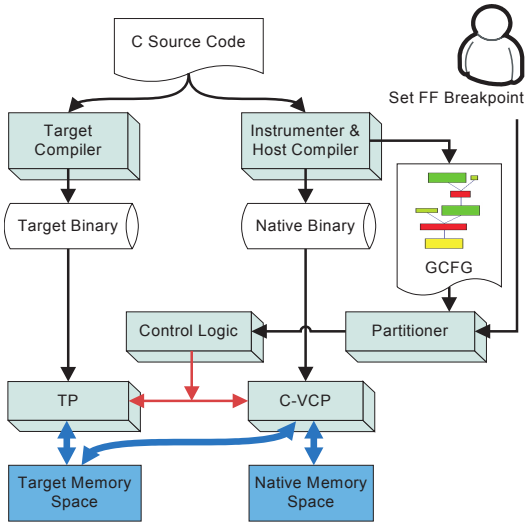


Figura 32: Fluxo de desenvolvimento de Kraemer et al. (7)

Uma vez gerados os binários alvo e nativos, juntamente com a instrumentação automática e estática do GCFG, é necessária a definição do particionamento para definir a lógica de controle (Control Logic), que dará acesso exclusivo ao software executando no processador alvo ou Target Processor (TP) e no Coprocessador Virtual C ou C-Virtual Coprocessor (C-VCP). Esta exclusão mútua, garante o acesso a memória alvo (Target Memory Space), para acesso de variáveis globais e ponteiros, sem necessidade de sincronismo. Existe também o ajuste do ponto de parada rápida ou FF Breakpoint, que é definido como um ponto do programa que deve ser atingido o mais rápido possível durante da definição do particionamento, como forma de melhorar o processo, sendo definido pelo projetista.

#### 3.2.1.2 Resultados

Na validação dos conceitos propostos foram utilizadas 4 aplicações de referência: DES (criptografia simétrica), JPEG (decodificador de imagem), MD5 (digestor de mensagem) e Susan (detecção de borda). A plataforma alvo considerada é a MIPS32 com modelo ISS e compilador alvo GCC versão 2.96 e plataforma nativa PC (Athlon X2 4600+) com 4 GB de memória, rodando sistema operacional Linux Fedora Core 4 e utilizando o compilador nativo GCC versão 4.0.2.

Tabela 8: Comparativo de ISS e HySim de Kraemer et al. (7)

| Aplicação | MIPS ISS | HySim | Desempenho | Erro |
|---|---|---|---|---|
| DES | 50,48s | 0,69s | 72x | 2,45% |
| JPEG | 40,57s | 5,71s | 7,1x | 17,62% |
| MD5 | 21s | 0,939s | 22,34x | 14,61% |
| Susan | 184s | 5,68s | 32,4x | 3,27% |

Nos experimentos realizados ficam evidentes taxas de melhoria de desempenho, que variam de cerca de 7 até 72 vezes com erro de 9,5%, quando se comparando a execução ISS. A aplicação JPEG teve o pior desempenho, e o motivo apresentado foi o uso de ponteiros de função que tornam um bom particionamento mais difícil, reduzindo significativamente a quantidade de software que executa nativamente no VCP.

#### 3.2.1.3 Contextualização

No trabalho de Kraemer et al. (7), é buscada a combinação da precisão dos modelos ISS com o alto desempenho dos modelos nativos de simulação. A abordagem proposta permite que um sistema possa ser executado em um ambiente híbrido (ISS + nativo), onde a parte dependente da plataforma executa no ISS e a parte independente executa nativamente.

Considerando os objetivos propostos, a abordagem de Kraemer et al. (7) permite um ambiente de simulação com suporte total ao desenvolvimento de HdS, entretanto, por usar ISS o desempenho desta parte de código ainda continua sendo um gargalo nas simulações. É interessante também abstrair a simulação da camada de software dependente do hardware, como forma de tornar seu desenvolvimento mais eficiente e sem dependência de utilização

de modelos ISS.

### 3.2.2 Combination of Instruction Set Simulator and Abstract RTOS Model Execution for Fast and Accurate Target Software Evaluation (Krause et al. (8), ACM 2008)

Para contornar o gargalo de simulação de sistemas complexos, o trabalho de Krause et al. (8) propõe a utilização de um modelo de RTOS em SystemC integrado ao ISS, de tal forma que o software executa no ISS integrado com o modelo de RTOS, descrito em nível de sistema. Desta forma, ocorre uma drástica redução no tempo de simulação, uma vez que todas as funções do RTOS são executadas pelo sistema nativo, deixando para o ISS apenas as funções da aplicação propriamente ditas.

#### 3.2.2.1 Fluxo de Desenvolvimento

O fluxo de desenvolvimento consiste em combinar o modelo de RTOS com ISS em modelo em nível de transação ou Transaction Level Model (TLM), como pode ser visto na figura 33, terceirizando todas as funções de escalonamento para o modelo de RTOS em SystemC, de mais alto nível e execução mais rápida.

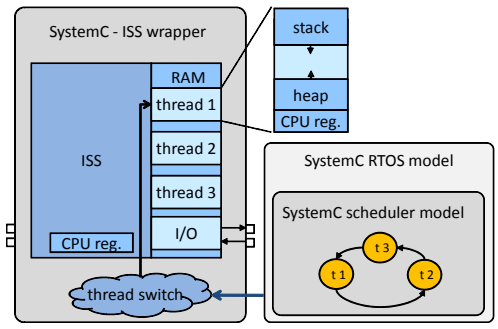


Figura 33: Arquitetura proposta de Krause et al. (8)

Ambos os ambientes de simulação, o ISS e o modelo RTOS, efetuam a troca de dados, como informações sobre escalonamento de tarefas (thread switch) e sua sincronização. Cada aplicação possui sua memória e registradores separados, permitindo que durante a troca de tarefa seja necessário somente trocar os ponteiros (stack, heap, CPU reg).

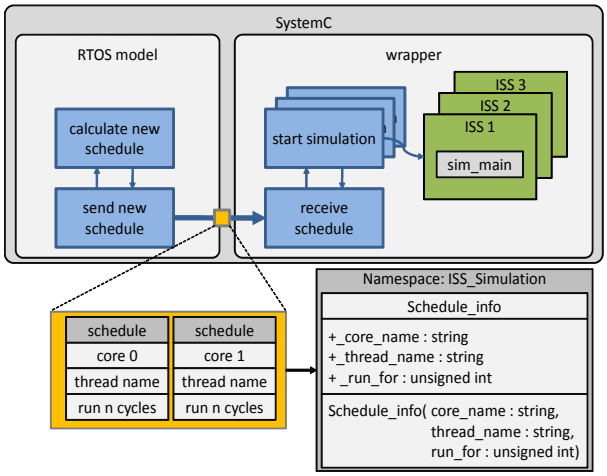


Figura 34: Modelo de simulação de Krause et al. (8)

No modelo de simulação completo, detalhado na figura 34, toda a plataforma em SystemC pode ser observada, com seu modelo de RTOS (RTOS model) e o envoltório para o ISS (wrapper), além das estruturas de dados de comunicação entre eles, contendo as informações de escalonamento para cada núcleo de processamento utilizado.

#### 3.2.2.2 Resultados

Para avaliar a proposta híbrida, foram utilizados o ISS Simplescalar, com conjunto de instruções ARM e o sistema operacional de tempo real RTEMS. Nas análises teóricas, a melhoria que desempenho ($MD$) é calculada pela proporção de tempo de simulação do RTOS no ISS ($TS_{RTOS}$) dividido pelo tempo de simulação do modelo RTOS proposto ($TS_{MODELO}$), como pode ser visto na equação 3.1. O $TS_{RTOS}$ é definido pela soma do tempo de execução do programa ($P$), do tempo ocioso ($O$) e do tempo de execução do RTOS ($E_{RTOS}$), já o $TS_{MODELO}$ é a soma do tempo do programa ($P$) e do tempo de execução do modelo do RTOS ($E_{MODELO}$).

$$MD = \frac{TS_{RTOS}}{TS_{MODELO}} = \frac{P + O + E_{RTOS}}{P + E_{MODELO}} \tag{3.1}$$

Nos resultados experimentais, a precisão foi medida utilizando ciclos de relógio, utilizando a execução totalmente baseada em ISS como referência, e as políticas de escalonamento usadas foram: round robin (RR) e priority-based rate monotonic (RMS). Foram utilizadas duas aplicações com as duas políticas

de escalonamento, sendo verificado o comportamento dos simuladores de acordo com o número de troca de tarefas realizadas.

Tabela 9: Desempenho e precisão de Krause et al. (8)

| Simplescalar ISS | Modelo RTOS | Desempenho | Erro |
|---|---|---|---|
| 9129936 | 9130679 | 1,8836x | 0,0081% |
| 9152704 | 9153444 | 1,8876x | 0,0081% |
| 9390271 | 9389517 | 1,9364x | 0,0080% |
| 11778182 | 11764153 | 2,4010x | 0,1191% |
| 22395341 | 22337821 | 4,3403x | 0,2568% |

Os resultados obtidos na tabela 9, tem os tempos de simulação calculados em número de ciclos de relógio, dos quais serão feitos o cálculo da melhoria de desempenho e da taxa de erro obtidas. Quanto mais troca de tarefas ocorrem, mais o sistema operacional é requisitado e consequentemente consome mais tempo de simulação, sendo por isto que quando mais ciclos de simulação são realizados, maior é a melhoria de desempenho oferecida.

#### 3.2.2.3 Contextualização

Com a forte motivação de melhorar o desempenho de simulação de sistemas, o trabalho de Krause et al. (8) segue uma linha híbrida de simulação combinando o tradicional modelo ISS com um modelo abstrato de RTOS, integrados em um ambiente virtual de simulação. Em sua abordagem o modelo ISS possui uma interface para o modelo de RTOS, assim delegando todo o comportamento referente ao SO para o modelo e desta forma reduzindo a quantidade de código executado no ISS.

Dentre os objetivos listados, o suporte para HdS é efetivo pelo uso de modelos baseados em ISS, mas o desempenho alcançado por Krause et al. (8) está abaixo do que se espera atingir em uma simulação nativa. Apesar de abstrair parte significativa do software do ISS, o trabalho de Krause et al. (8) ainda concentra no ISS todo o HdS e suas funcionalidades, conseguindo excelentes taxas de erro, mas com melhorias de desempenho modestas.

## 3.3 Geração de Software da Síntese de Modelos

Neste paradigma, a estratégia adotada busca a criação de modelos executáveis com alto nível de abstração, utilizando linguagens de projeto em nível de sistema ou System Level Design Language (SLDL) para implementar o comportamento esperado. Existem algumas opções de linguagens disponíveis, normalmente focadas em resolver problemas específicos, mas para fins de exemplificação, pode-se citar a linguagem SystemC (31), que permite diversas modelagens para simulação de sistemas e plataformas.

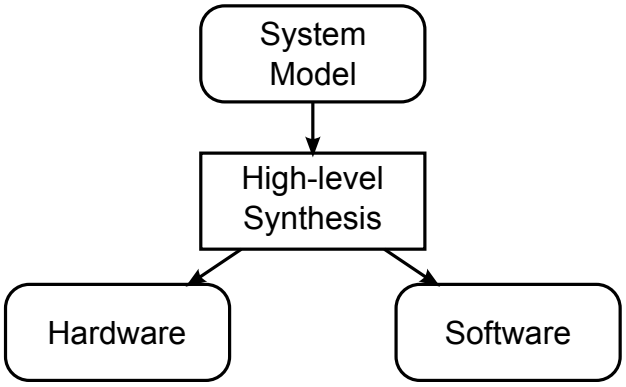


Figura 35: Fluxo de geração de software

Para permitir o uso destes modelos, são criadas ferramentas que permitem o seu refinamento em implementações de hardware e software. Cada trabalho possui suas peculiaridades, mas de maneira genérica, um compilador de sistema é construído para processar estas descrições criadas em SLDL, realizando passos intermediários, que são realizados por ferramentas (fluxo simplificado na figura 35), e, por fim, o software ou parte dele é gerado automaticamente, refletindo o comportamento de sua especificação de alto nível.

### 3.3.1 A Design Flow Based on Domain Specific Language to Concurrent Development of Device Drivers and Device Controller Simulation Models (Lisboa et al. (9), ACM 2009)

A abordagem do trabalho de Lisboa et al. (9) é motivada pela necessidade dos sistemas embarcados de se comunicarem com diferentes dispositivos, dependendo sempre de estruturas de hardware e software para concretizar esta comunicação. São feitas duas categorizações: uma para o lado do

software, chamado de gerenciador de dispositivo ou device driver, que fornece acesso transparente para as funções do dispositivo; e do lado do hardware, o controlador do dispositivo ou device controller, que implementa o protocolo associado no próprio dispositivo de hardware para permitir o acesso do software de gerenciamento.

#### 3.3.1.1 Fluxo de Desenvolvimento

Existem duas visões claras na abordagem de Lisboa et al. (9): a do controlador ou controller, que é implementado em hardware e efetivamente controlará o dispositivo em questão, atendendo a um protocolo de comunicação com o processador; e a do gerenciador ou driver, que é a implementação em software que gerencia este controlador, através desta interface com o controlador, permitindo a aplicação acesso transparente as suas funcionalidades.

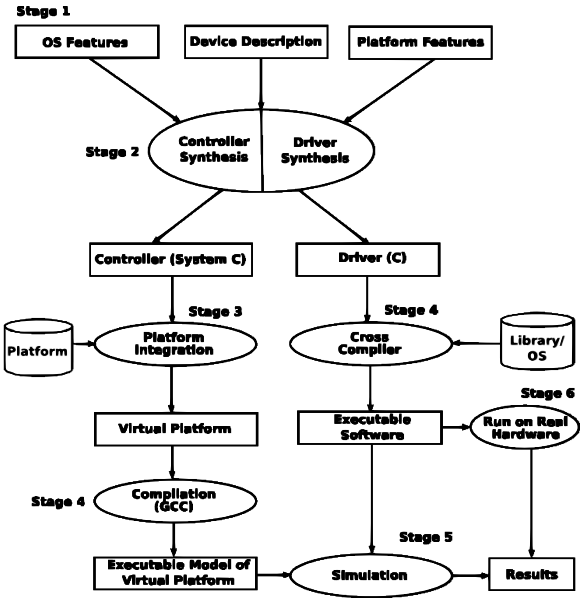


Figura 36: Fluxo de Desenvolvimento de Lisboa et al. (9)

O desenvolvimento se inicia pela especificação de todas as funcionalidades na linguagem DevC, chamado de estágio 1, que pode ser visualizado na figura 36, definindo as funcionalidades disponíveis do sistema operacional (OS Features); a descrição do dispositivo em alto nível (Device Description); e as funções da plataforma (Platform Features). No estágio 2, com as descrições dos modelos de controlador e gerenciador de dispositivos, é realizada a síntese de cada um deles, para que seus modelos sejam gerados. Como o controlador é um componente de hardware, seu modelo é gerado na linguagem

SystemC, que permite o seu uso em plataformas virtuais, inclusive com integração com modelos de implementação reais. Com os modelos gerados e todos os ajustes necessários terem sido realizados, tem início o estágio 3, que realiza a integração do controlador na plataforma virtual alvo, adicionando componentes e envoltórios que facilitem ou permitam a utilização deste componente na plataforma. O software em C gerado e a plataforma virtual são compilados para simulação, no estágio 4, utilizando as dependências necessárias.

Com as versões executáveis, tanto da plataforma, como do software embarcado, tem início o estágio 5 que realiza as simulações destes componentes integrados, obtendo uma série de resultados que serão exibidos ou armazenados para comparação e validação posterior, utilizando o dispositivo real em hardware. No estágio final 6, a implementação real, rodando em tecnologia FPGA ou ASIC, é executada e os mesmos resultados do estágio 5 devem ser obtidos, permitindo a comparação e a validação do conjunto hardware e software criado.

#### 3.3.1.2 Resultados

Na avaliação da eficácia da abordagem proposta, foi escolhido um estudo de caso para controlar uma tela de cristal líquido gráfica (módulo PCF8833), onde duas frentes seriam exploradas: uma com a implementação completamente manual, seguindo a abordagem tradicional e outra utilizando as ferramentas e linguagens propostas para o desenvolvimento.

Tabela 10: Comparativo do número de linhas de Lisboa et al. (9)

| Linhas sintetizadas | | Linhas desenvolvidas | | Percentual de síntese | |
|---|---|---|---|---|---|
| Hardware | Software | Hardware | Software | Hardware | Software |
| 72 | 36 | 16 | 2 | 81% | 94% |
| 100 | 58 | 15 | 3 | 86% | 95% |
| 85 | 76 | 25 | 16 | 40% | 88% |
| 45 | 26 | 132 | 4 | 25% | 87% |
| 61 | 46 | 132 | 4 | 32% | 92% |

Na tabela 10, são comparadas o número de linhas sintetizadas pela abordagem proposta e o número de linhas desenvolvidas quando a estratégia tradicional é utilizada. Cada linha representa um modelo diferente de dispositivo baseado em caractere utilizado, com linhas de código para o controlador em hardware e para o gerenciador em software.

#### 3.3.1.3 Contextualização

O trabalho de Lisboa et al. (9) procura melhorar o desenvolvimento de software embarcado, especialmente o HdS, através da definição de uma DSL que permite a modelagem da comunicação entre o sistema e o dispositivo da plataforma. Apesar de seu caráter restrito para HdS, o trabalho de Lisboa et al. (9) focou na necessidade de uso de modelos mais abstratos para melhorar o fluxo de desenvolvimento, com a proposição de uma DSL.

Considerando os objetivos propostos, apesar do foco em HdS, a abordagem de Lisboa et al. (9) não oferece suporte ao desenvolvimento do comportamento do software em si, concentrando-se na geração de interfaces de software e controladores de hardware. Outro ponto pertinente é que as simulações dependem de ISS para serem feitas, recaindo em baixos desempenhos simulação que podem inviabilizar o desenvolvimento de sistemas mais complexos.

### 3.3.2 Fast and Accurate Processor Models for Efficient MPSoC Design (Schirner et al. (10, 11), ACM 2010)

A abordagem de Schirner et al. (10, 11) observa que toda esta quantidade crescente de funcionalidades no software representa um grande desafio e que utilizar a abordagem tradicional baseada em ISS não é mais uma opção viável. Existe uma necessidade evidente de se ter modelos mais flexíveis e mais rápidos, capazes de atender melhor o fluxo de desenvolvimento de plataformas multiprocessadas.

#### 3.3.2.1 Fluxo de Desenvolvimento

Inicialmente, não existe informação de tempo e nem detalhes de implementação na modelagem, com todo o comportamento sendo agrupado em processos que se comunicam por canais ponto a ponto de diversos tipos, possibilitando comunicação síncrona, assíncrona, bloqueante ou não bloqueante. Após a construção e avaliação desta estrutura inicial, é preciso que o projetista tome as decisões arquiteturais necessárias, como: número de unidades de processamento ou de componentes de hardware e como o com-

portamento será mapeado em cada uma destas unidades. Com estas duas entradas, a definição da plataforma em SpecC, com os comportamentos e a tomada de decisões arquiteturais, o compilador de sistema é capaz de gerar um sistema TLM ou BFM, como pode ser visto na figura 37.

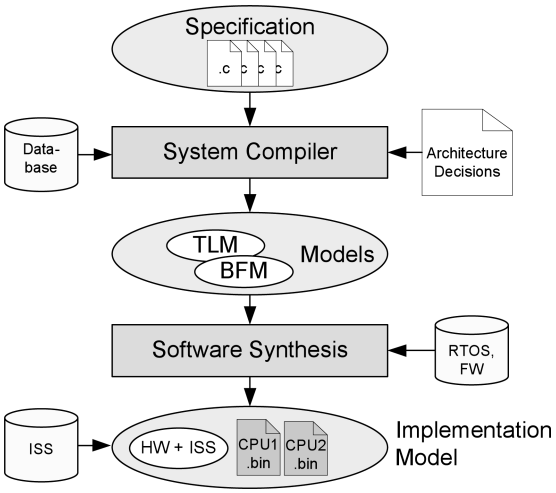


Figura 37: Fluxo de Desenvolvimento de Software de Schirner et al. (10, 11)

O sistema TLM ou BFM gerado reflete todo o comportamento e decisões de projeto tomadas, permitindo que novas explorações sejam feitas, o desempenho seja avaliado e que a depuração possa ser realizada. Focando na síntese de software, seguindo o fluxo descrito na figura 37, tudo tem início no modelo TLM ou BFM que como já foi dito, reflete todo o comportamento e decisões de projeto realizadas: número de processadores, políticas de escalonamento e mapeamento das tarefas. Dois passos são realizados na síntese de software: a geração de código e síntese de HdS. Na geração de código, é feita a tradução da especificação do software em SpecC para código C, resolvendo as estruturas hierárquicas, portas e conexões de canais. Na síntese de HdS, códigos para comunicação interna e externa do processador são gerados, incluindo gerenciadores de dispositivos e sincronização, seja por polling ou interrupção.

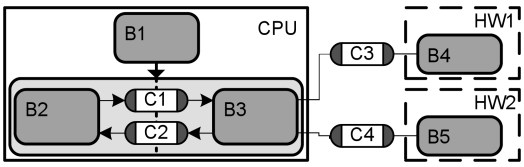


Figura 38: Modelo de Aplicação de Schirner et al. (10, 11)

Tomando um exemplo de sistema, ilustrado na figura 38, está um conjunto de funções B1, B2 e B3 executando no processador, se comunicando pelos

canais C1 e C2, e as funções B4 e B5 sendo desempenhadas por componentes de hardware dedicados, conectados pelos canais de comunicação C3 e C4. Esta especificação possui também as definições de hierarquia dos processos, canais e dos componentes de hardware, estruturando cada nível como uma camada diferente, responsável pelo processamento (CPU).

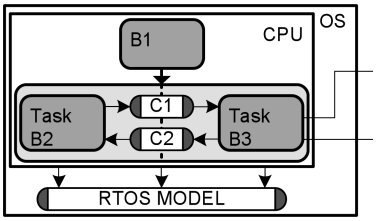


Figura 39: Modelo de Tarefa de Schirner et al. (10, 11)

O escalonamento das tarefas de software B1, B2 e B3, sua priorização e sincronismo são realizados pela camada de sistema operacional (OS), como pode ser visto na figura 39. Um modelo de RTOS é fornecido, permitindo análise de políticas de escalonamento, impacto da priorização das tarefas e mecanismos de controle de fluxo para execução sequencial e paralela.

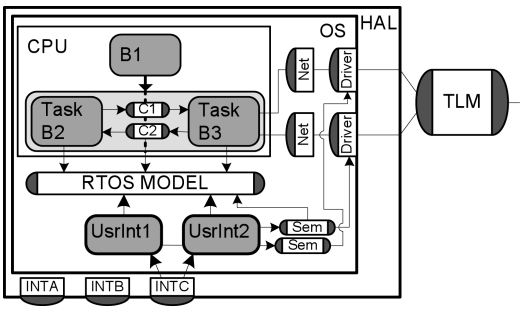


Figura 40: Modelo de Programa de Schirner et al. (10, 11)

No modelo de gerenciamento de dispositivo ou Firmware (FW), ilustrado na figura 40, o acesso aos componentes de hardware da plataforma é modelado pela camada de abstração de hardware ou Hardware Abstraction Layer (HAL).

Para oferecer um modelo de processador, conforme observado na figura 41, um gerenciador de interrupções de hardware é inserido e uma nova camada de núcleo de processamento (Core) é adicionada. O dispositivo de interrupções de hardware está conectado a um controlador programável de interrupção ou Programmable Interruption Controller (PIC) que, juntamente com a interface de comunicação TLM, o conecta aos demais componentes.

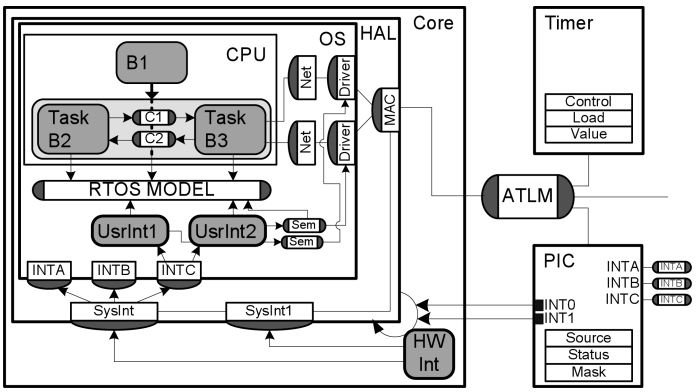


Figura 41: Modelo de Processador de Schirner et al. (10, 11)

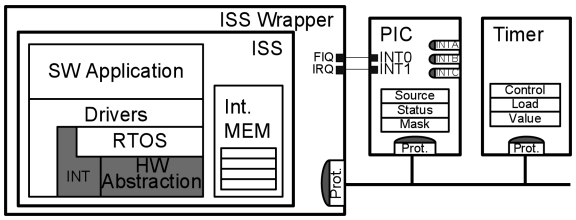


Figura 42: Modelo de BFM de Schirner et al. (10, 11)

No nível BFM, uma interface de barramento precisa deve ser suportada, seja com precisão de pinos ou de ciclos. O comportamento é idêntico ao obtido nos modelos TLM, mas com refinamentos que detalham a comunicação de tal forma que componentes de implementação de hardware (RTL) possam ser conectados e simulados em conjunto, para depuração e análise de resultados.

#### 3.3.2.2 Resultados

Para avaliação da proposta, foi utilizada uma plataforma multiprocessada, conforme pode ser observado na figura 43. São executadas nesta plataforma 3 aplicações industriais: GSM 06.60 transcodificador de voz, um decodificador de áudio no formato MP3 e um codificador de imagem no padrão JPEG. Estas funcionalidades foram mapeadas nesta plataforma, com implementações de hardware e software combinadas, sendo desempenhadas pelos processadores ARM7TDMI e DSP 5660k executando a 100 e 60 MHz, respectivamente.

Os resultados foram gerados considerando duas configurações: a primeira onde todas as funções são implementadas em software e a segunda em que as partes de computação intensivas são implementadas em componentes de hardware dedicados. O desempenho e a precisão de simulação destas duas

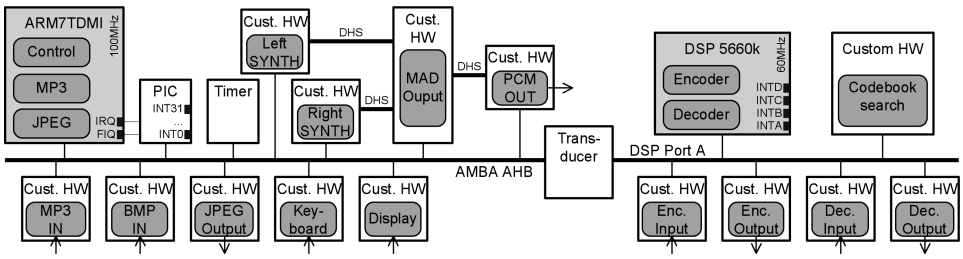


Figura 43: Plataforma Multiprocessada de Schirner et al. (10, 11)

configurações serão sumarizadas, contendo a melhoria de desempenho obtida, o erro relativo ao modelo ISS e quantos mega ciclos por segundo (MCiclos/s) foram simulados em cada tipo de modelo.

Tabela 11: Simulação de software de Schirner et al. (10, 11)

| Referência | | Aplicação | Tarefa | FW | TLM | BFM | ISS |
|---|---|---|---|---|---|---|---|
| Desempenho | MP3 | 43563 | 32672 | 30159 | 6427 | 80 | 1 |
| | JPEG | 8022 | 35653 | 32088 | 16044 | 66 | 1 |
| | GSM | 54762 | 54762 | 54762 | 51229 | 6456 | 1 |
| Erro | MP3 | 46,19% | 1,51% | 1,51% | 1,63% | 1,63% | 0% |
| | JPEG | 58,13% | 3,64% | 3,61% | 2,32% | 2,32% | 0% |
| | GSM | 6,78% | 1,13% | 1,10% | 0,72% | 0,72% | 0% |
| MCiclos/s | MP3 | 2080 | 1560 | 1440 | 307 | 3,8 | 0,05 |
| | JPEG | 318 | 1415 | 1273 | 637 | 2,6 | 0,04 |
| | GSM | 411 | 411 | 411 | 384 | 48,4 | 0,01 |

Na tabela 11, os dados de simulação para uma configuração somente de software são organizados, sob perspectiva de desempenho, erro e MCiclos por segundo, para cada uma das 3 aplicações utilizadas que são o MP3, JPEG e GSM.

Tabela 12: Simulação de hardware/software de Schirner et al. (10, 11)

| Referência | | Aplicação | Tarefa | FW | TLM | BFM | ISS |
|---|---|---|---|---|---|---|---|
| Desempenho | MP3 | 13835 | 12577 | 12577 | 3739 | 8 | 1 |
| | MP3+JPEG | 8797 | 21992 | 21992 | 7331 | 20 | 1 |
| | GSM | 32658 | 31570 | 30551 | 15785 | 1627 | 1 |
| Erro | MP3 | 45,72% | 21,11% | 13,02% | 1,17% | 1,17% | 0% |
| | MP3+JPEG | 56,06% | 14,90% | 10,64% | 3,09% | 3,09% | 0% |
| | GSM | 16,37% | 3,56% | 3,53% | 1,68% | 1,68% | 0% |
| MCiclos/s | MP3 | 1440 | 1309 | 1309 | 389 | 0,8 | 0,10 |
| | MP3+JPEG | 367 | 916 | 916 | 305 | 0,8 | 0,04 |
| | GSM | 341 | 330 | 319 | 165 | 17 | 0,02 |

As aplicações podem se beneficiar enormemente com o uso de hardware dedicado, principalmente para aquelas operações de computação intensiva

e com grande consumo de dados. É neste cenário que foi feita a análise de desempenho, erro e MCiclos por segundo, como visto na tabela 12, para as mesmas aplicações MP3, MP3+JPEG e GSM, com suporte de hardware.

#### 3.3.2.3 Contextualização

No trabalho de Schirner et al. (10, 11) é proposta uma abordagem de refinamento de um sistema em alto nível, através de uma série de níveis intermediários, até a implementação em código fonte do software. A motivação principal para este esforço está na crescente necessidade por modelos de simulação mais rápidos e flexíveis, além de suportar as arquiteturas multiprocessadas. Em cada refinamento são adicionados mais detalhes comportamentais e estruturais ao modelo do sistema, permitindo inclusive que o HdS seja gerado, em forma de gerenciadores de dispositivos, de interrupções e camada de abstração de hardware.

Considerando os objetivos listados, a proposta de Schirner et al. (10, 11) resolve com sucesso o problema de abstrair a complexidade do modelo de simulação de software embarcado, suportando inclusive o HdS. Porém, esta abordagem não apresenta um ambiente de execução de alto nível para a implementação de software gerada, recaindo no baixo desempenho do ISS. A questão chave destes requisitos é permitir que o projetista fosse capaz de desenvolver seu sistema como se estivesse utilizando um ISS, com alto nível de detalhes e com alto desempenho, mas sem a necessidade de domínio de técnicas de modelagem ou de ferramentas de síntese de sistemas.

## 3.4 Análise Comparativa

Uma vez que todos os trabalhos relacionados foram classificados e detalhados, é interessante que seja feita uma análise comparativa e sistemática dos trabalhos pesquisados com os objetivos propostos neste trabalho. Desta forma, espera-se que o leitor tenha uma visão clara das lacunas existentes no estado da arte considerado e como este trabalho procura contribuir no suporte ao fluxo de desenvolvimento de sistemas.

Na tabela 13, todos os trabalhos considerados foram analisados sob os se-

Tabela 13: Análise comparativa dos trabalhos

| Trabalhos | Desenvolvimento de HdS | Desempenho de Simulação | Erro das Estimativas |
|---|---|---|---|
| Ecker et al. (1) | Sim | 250x-2x | - |
| Wang et al. (2) | Parcial | 400x | 2,21%-0,16% |
| Lo et al. (3) | Sim | 60x-38x | 0% |
| Razaghi et el (4) | - | 6.862x-4.262x | 4,18% |
| Gerstlauer et al. (5) | - | 2.400x | 10,3%-0,0004% |
| Yeh et al. (6) | Sim | - | - |
| Kraemer et al. (7) | - | 72x-7x | 18%-2% |
| Krause et al. (8) | - | 4x-2x | 0,26%-0,01% |
| Lisboa et al. (9) | Sim | - | 0% |
| Schirner et al. (10, 11) | Sim | 14.568x-2x | 42%-3% |

guintes aspectos: o desenvolvimento de HdS, ou seja, se o modelo ou ferramentas levam em consideração o desenvolvimento de software dependente do hardware; o desempenho de simulação, que define o aumento no desempenho de simulação, normalmente tendo como referência modelos ISS; e o erro das estimativas, que mede qual a precisão das estimativas de tempo simulado do trabalho proposto em relação aos modelos baseados em ISS ou em implementação RTL com precisão de ciclo.

Para estabelecer uma comparação quantitativa entre os trabalhos relacionados no estado da arte e este trabalho proposto, os experimentos realizados utilizam a mesma metodologia de comparação dos resultados de melhoria de desempenho de simulação e de erro de estimativas, utilizando um modelo com precisão de instrução como referência. Apesar de utilizarem estudos de casos e níveis de modelagem distintos, os resultados mostrados fornecem métricas que permitem a comparação entre as diferentes abordagens. Durante a descrição dos experimentos é feita a comparação com os trabalhos relacionados e com resultados obtidos de plataformas de hardware, com o objetivo de destacar a consistência dos dados obtidos, a melhoria de desempenho de simulação e a taxa de erro das estimativas geradas.

Pelos trabalhos relacionados neste capítulo e pelos objetivos definidos para este trabalho proposto na seção 1.2, é esperado que o leitor perceba o cenários das abordagens baseadas em simulação nativa, no qual este trabalho proposto está inserido, e quais as suas contribuições para o estado da arte

considerado. Para evitar redundância no texto, as publicações geradas por esta abordagem proposta são apenas referenciadas (57, 58).

# 4 Modelagem Proposta

Este capítulo é dedicado a detalhar o mecanismo de modelagem proposto para simulação nativa de sistemas, considerando os conceitos, as motivações e objetivos já explicitados nos capítulos anteriores. Além disto, será sempre oferecida ao leitor uma visão paralela com o estado da arte considerado, como forma de evidenciar a contribuição da abordagem proposta.

No contexto de desenvolvimento de sistemas existe um compromisso entre os aspectos de precisão (accuracy) e desempenho de execução (performance) do modelo, como é bem ilustrado na figura 44. Observando a tradicional estratégia de simulação baseada em ISS, é constatado que ela apresenta um alto nível de precisão, comparável à implementação em circuitos integrados do hardware (Real HW), mas todo este nível de detalhamento implica em um baixo desempenho de simulação. Já no extremo oposto, é visto que o código da aplicação (App Code), executando nativamente com alto desempenho, também comparável ao dispositivo real, entretanto gerando estimativas com baixa precisão, devido ao baixo nível de detalhe da plataforma alvo na execução nativa.

O trabalho proposto consiste no desenvolvimento de técnicas que permitam a implementação de um modelo rápido (Fast Model) para execução de software que seja capaz de oferecer altos níveis de desempenho e precisão de simulação. A ideia principal consiste em abstrair a grande quantidade de detalhes funcionais do modelo ISS, porém procurando manter níveis de precisão elevados e baixas taxas de erro nas estimativas de tempo simulado geradas.

Nas próximas seções deste capítulo, todas as técnicas e mecanismos propostos serão detalhados, partindo de uma visão geral que irá contextualizar a modelagem proposta com os diversos níveis de abstração possíveis, passando

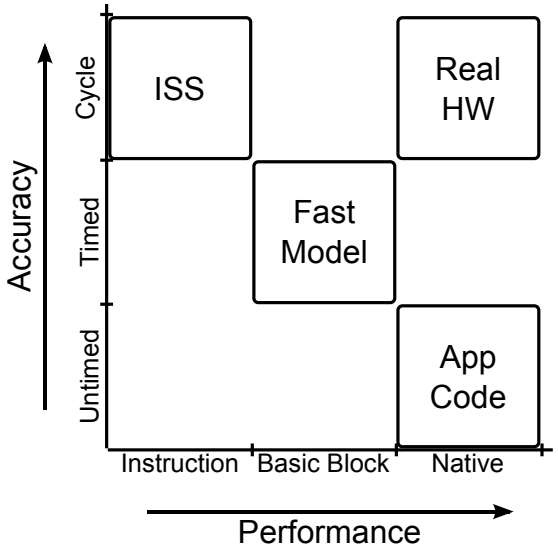


Figura 44: Compromisso de precisão de simulação versus desempenho

pela organização da modelagem proposta, seus princípios de funcionamento e, por fim, uma descrição de como este trabalho suporta o paradigma de sistemas com vários elementos de processamento (MPSoC) (44).

## 4.1 Visão Geral

Uma visão ampla dos vários níveis de abstração na especificação de sistemas é muito bem ilustrado em Schirner et al. (10, 11), partindo de um modelo de aplicação de alto nível e, através de refinamentos estruturais e comportamentais, sendo feita a geração, correta por construção, dos componentes de hardware e software da plataforma final. A grande vantagem deste desenvolvimento de cima para baixo, além das já mencionadas anteriormente, é de não atrelar previamente o comportamento do sistema com sua implementação final. Esta independência tecnológica permite mais flexibilidade para exploração do espaço de projeto, possibilitando uma melhor avaliação das alternativas de implementação e de mapeamento dos recursos.

O fluxo de desenvolvimento em nível de sistema, com os seus diferentes níveis de abstração, pode ser visto na figura 45, tendo início no nível mais alto de abstração de aplicação (Application) até o nível mais detalhado baseado em simuladores de conjunto de instrução (ISS). Por trabalhar no nível mais alto de abstração, a implementação do modelo de sistema em nível de aplicação está restrita ao domínio de linguagens de descrição em nível de sistema (SLDL),

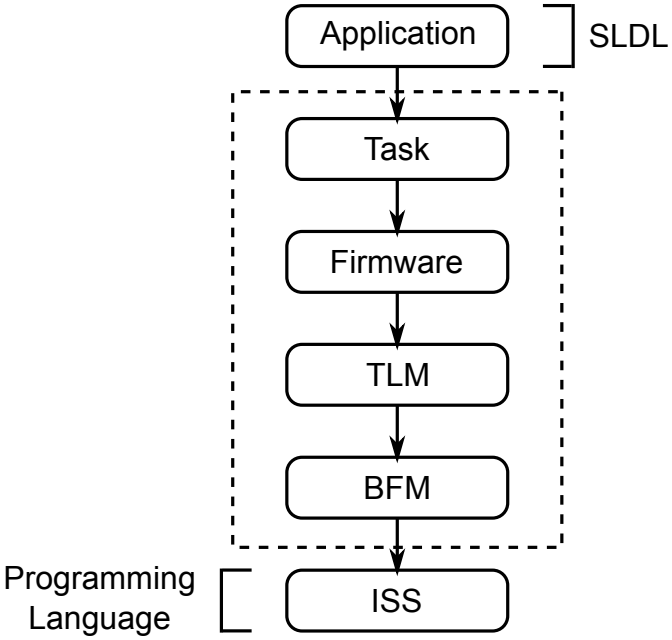


Figura 45: Abstração no fluxo de desenvolvimento em nível de sistema

uma vez que fornecem toda a infraestrutura necessária para simulação dos comportamentos e das estruturas do sistema. No extremo oposto do fluxo, no nível mais baixo de abstração, é utilizado um modelo ISS que executa a código binário resultante da compilação do software descrito em uma linguagem de programação (Programming Language), como ANSI C (59).

A modelagem proposta está situada nos níveis de tarefa (Task), de gerenciamento de hardware (Firmware), nível de transação (TLM) e funcional de barramento (BFM), ilustradas na figura 45 pela área destacada em padrão tracejado. Nestes níveis de abstração é permitida a utilização de componentes implementados tanto em SLDL como em linguagens de programação, entretanto a modelagem proposta tem como requisito a execução da implementação do software em linguagem de programação, ao invés de um modelo do sistema, permitindo a avaliação mais realista do sistema em uma determinada configuração de plataforma alvo.

Ainda referenciando a figura 45, existem diferentes níveis de abstração em que a modelagem proposta pode ser aplicada, mas para fins de comparação de resultados e para atender os requisitos elicitados, foi escolhida a modelagem em nível de transação ou TLM (26) para descrição da modelagem e para realização dos estudos de caso. Este nível de abstração permite a realização das operações de comunicação externa e a utilização de camadas de soft-

ware para suporte de SO (OS Support) e camada de abstração de hardware (HAL), como é ilustrado na figura 46. Neste nível de modelagem TLM é possível exercitar todas as funcionalidades do sistema referentes à interface com o hardware da plataforma, o que não possível em níveis mais elevados de abstração, que não modelam a comunicação externa e possui papel essencial para suportar esta abordagem proposta. Outro ponto importante é que permite que a comunicação ocorra através de transações, sem os detalhes excessivos de uma descrição BFM que causariam um impacto negativo do desempenho de simulação.

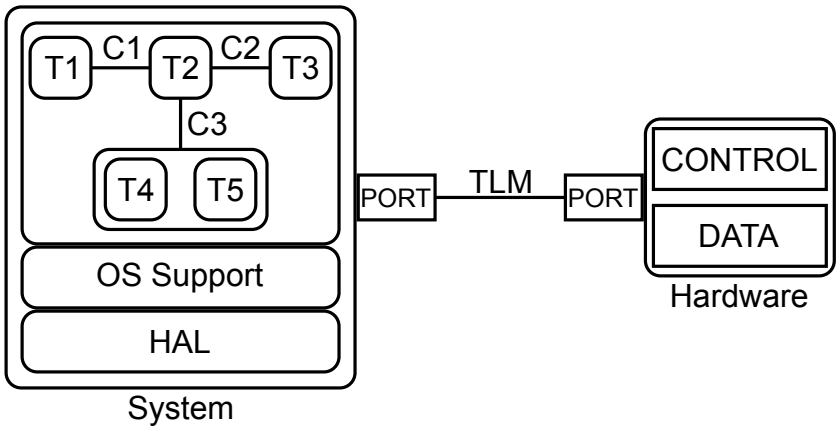


Figura 46: Modelo de sistema em nível de transação

A especificação de plataformas em nível TLM apresenta um dos melhores compromissos entre desempenho e precisão dentre todos os níveis vistos, sendo amplamente utilizados tanto pela indústria como pela academia. Ilustrado na figura 11, as funções do sistema operacional (OS Support) e da camada de abstração de hardware (HAL) permitem a utilização de recursos de multiprogramação e de acesso aos recursos de hardware da plataforma, respectivamente.

Por estas características, este nível de abstração é o mais adequado para demonstrar as capacidades da modelagem proposta principalmente com relação às interfaces de acesso, de controle e de interrupção dos dispositivos de hardware da plataforma. A grande contribuição da modelagem proposta é a integração da execução nativa de uma aplicação descrita em linguagem de programação com um ambiente de simulação com aproximação de tempo simulado, como se a aplicação estive executando em um modelo com precisão de instrução ou no hardware real.

## 4.2 Fluxo de Desenvolvimento Proposto

Considerando cada nível de abstração de sistema e suas características, o fluxo proposto é definido para suportar cada uma das características dos diferentes níveis de modelos. Desde o mais alto nível de modelo de aplicação até o modelo funcional de barramento este fluxo pode ser aplicado, bastando que seja definido qual o nível de abstração desejado. Um aspecto chave do fluxo proposto é possibilitar que o projetista seja capaz de modificar as decisões de projeto e rapidamente avaliar o impacto das escolhas realizadas.

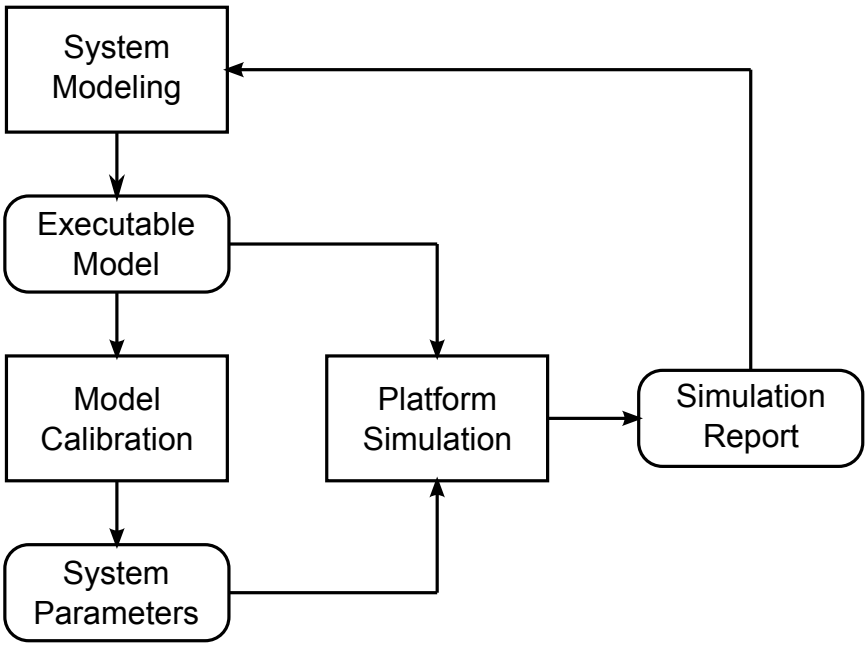


Figura 47: Fluxo de desenvolvimento proposto

No fluxo de desenvolvimento proposto, que pode ser visualizado na figura 47, são previstas três etapas principais:

- Modelagem do Sistema (System Modeling): é o processo de modelagem para capturar a especificação da plataforma e incorporar o código fonte da aplicação para ser executado na plataforma. Após a especificação de todos os modelos do sistema e da plataforma virtual, utilizando a SLDL SystemC, é feita a compilação nativa para que seja gerado um modelo executável da plataforma (Executable Model);

- Calibração do Modelo (Model Calibration): como a especificação do modelo proposto não possui informações sobre as restrições do sistema, é

preciso que o modelo desenvolvido seja calibrado para atender as especificações do projeto. Através da especificação das restrições temporais e do uso de ferramentas de desenvolvimento, o projetista do sistema consegue obter rapidamente os parâmetros que o modelo do sistema (System Parameters) precisa utilizar para que as restrições do projeto sejam satisfeitas. Este tópico será o tema principal do capítulo 5, onde serão fornecidos todos os detalhes sobre seu funcionamento e configuração;

- Simulação da Plataforma (Platform Simulation): utilizando o modelo executável da plataforma e os parâmetros de configuração obtidos na calibração, é feita a simulação da plataforma para que seja gerado um relatório de simulação (Simulation Report). Neste relatório estão contidas as informações relevantes para o projeto do sistema, como desempenho ou tempo de resposta, por exemplo. Com estas informações, é possibilitado ao projetista realizar ajustes na modelagem da plataforma para melhorar o desempenho ou reduzir o consumo de recursos.

Nas subseções a seguir será discutido com mais detalhes como cada uma destas etapas é organizada, explicitando quais as entradas e saídas necessárias, além dos artefatos gerados ao longo do processo de modelagem.

### 4.2.1 Modelagem do Sistema

Para realização da modelagem do sistema são necessárias informações sobre a plataforma de desenvolvimento (Platform Specification) e acesso ao código fonte da aplicação (Application Source Code) para sua execução nativa. A especificação da plataforma inclui características da unidade de processamento que se deseja modelar, sendo obrigatório ao projetista definir a frequência de operação, taxa de execução de instruções e tipo de interface com barramento para comunicação ou interrupção.

O fluxo de modelagem do sistema é ilustrado na figura 48, iniciando com as informações da plataforma e o código fonte da aplicação e terminando com um modelo executável da plataforma. Para contemplar esta fase do desenvolvimento são necessárias duas etapas:

- Modelagem HdSC (HdSC Modeling): nesta etapa todas as decisões de

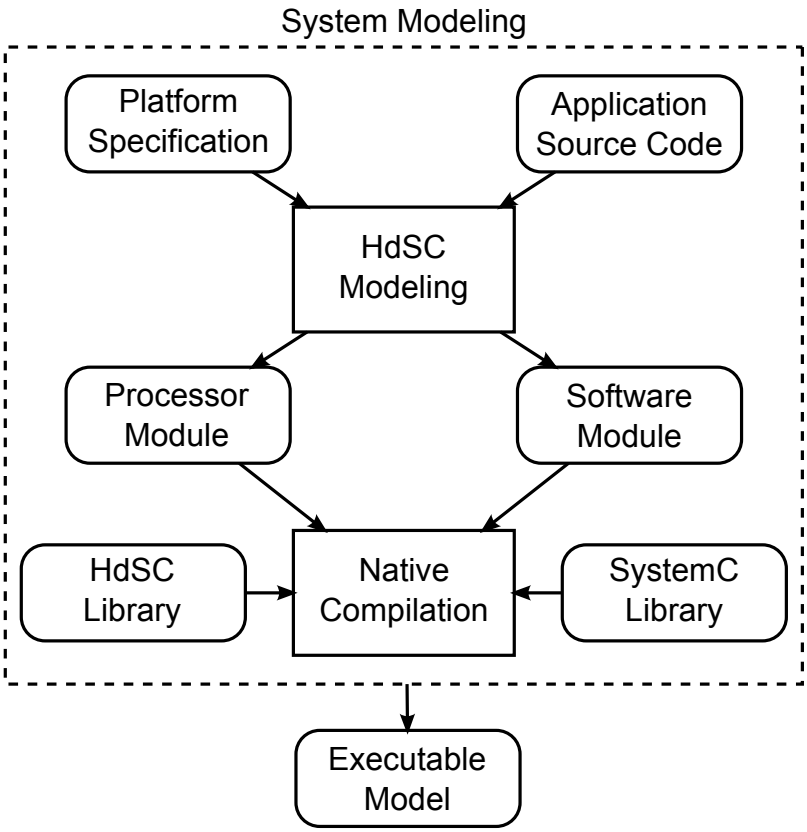


Figura 48: Fluxo de modelagem do sistema

projeto devem ser tomadas, incluindo a escolha de quais partes do sistema serão implementadas em software e quais serão em hardware. Com estas definições, é preciso especificar os módulos de processador e de software para execução e encapsulamento do código fonte da aplicação, respectivamente. A modelagem, utilizando a linguagem proposta HdSC, permite a abstração direta de um modelo ISS para o processador e de uma memória para o software da aplicação, suportando o desenvolvimento e a simulação de HdS;

- Módulo de Processador (Processor Module): possui o papel de processar o software da aplicação, permitindo a execução nativa e o acesso aos recursos da plataforma, como acesso aos dispositivos. Este módulo de processamento implementa os mecanismos de escalonamento, realiza estimativas de tempo simulado do software, modela a priorização e o tratamento de interrupções, permite o uso de protocolos de comunicação e suporta a especificação do comportamento de instruções e registradores de uma determinada arquitetura alvo;

  - Módulo de Software (Software Module): é o componente do sistema responsável pelo encapsulamento do código fonte da aplicação e pelo acesso aos recursos de hardware da plataforma, possuindo função estrutural análoga a uma memória que armazena o código binário da aplicação. Esta organização procura abstrair o papel que a memória possui em modelos de sistema mais precisos, como os baseados em ISS.

- Compilação Nativa (Native Compilation): com o suporte da biblioteca proposta (HdSC Library) e da SLDL (SystemC Library), é possível utilizar um compilador nativo para gerar um modelo executável (Executable Model) da plataforma, incluindo os componentes de hardware e de software integrados. Com este artefato, o projetista é capaz de avaliar o comportamento dinâmico do sistema e realizar os ajustes que se fizerem necessários.

Uma vez concluída a modelagem do sistema que é detalhada na seção 4.3, é necessária a calibração do modelo que será tema da próxima subseção. A etapa de calibração tem como objetivo determinar os parâmetros do sistema necessários para ajustar o modelo às restrições temporais do sistema.

### 4.2.2 Calibração do Modelo

Com os componentes do sistema modelados e uma organização de plataforma concebida, é necessário realizar a calibração do modelo de sistema especificado. A calibração é necessária pelo fato dos componentes modelados não incorporarem informações sobre a execução do sistema em alguma tecnologia ou plataforma alvo. Assim, para configurar o modelo abstrato desenvolvido, é preciso verificar as restrições temporais do projeto, como, por exemplo a frequência máxima de operação, e realizar simulações com o modelo executável desenvolvido, para que informações sobre a execução da plataforma sejam analisadas.

Na figura 49 o fluxo de calibração é ilustrado, iniciando com as restrições temporais (Timing Constraints) e com o modelo executável do sistema (Executable Model), os parâmetros de configuração do sistema (System Parameters) são gerados. Para calibrar o modelo do sistema são necessárias duas etapas:

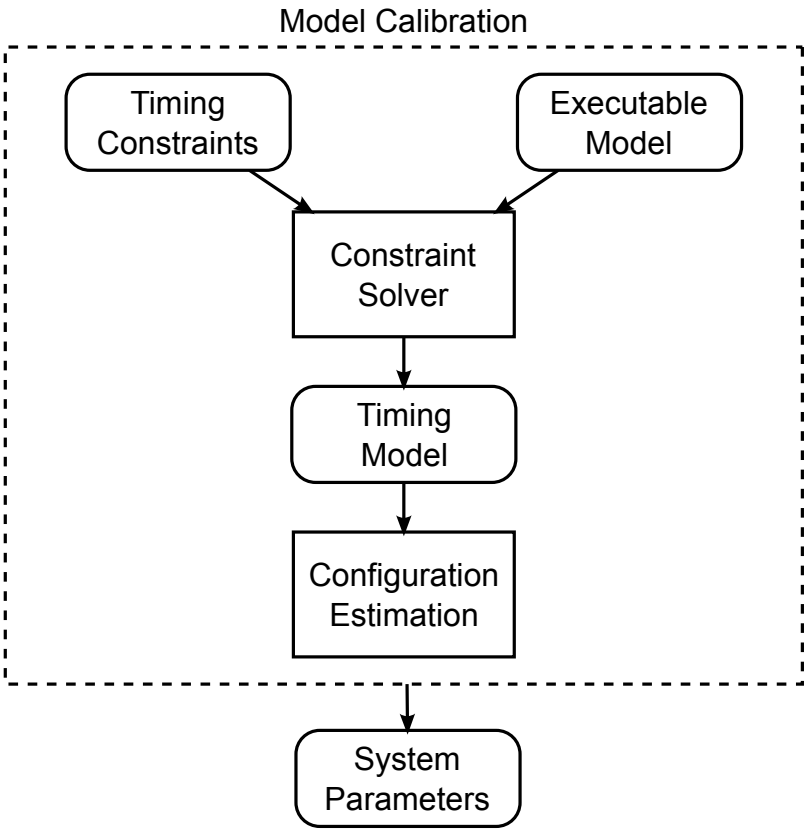


Figura 49: Fluxo de calibração do modelo

- Resolvedor de Restrições (Constraint Solver): utilizando algoritmos para gerar um modelo de tempo do sistema (Timing Model), através das restrições temporais e do modelo executável, esta etapa permite que o projetista obtenha automaticamente os parâmetros necessários para satisfazer as restrições impostas, caso exista uma solução viável. Este modelo de tempo simulado será detalhado nas próximas seções e sua caracterização permite que o comportamento do sistema seja previsto estaticamente, com a necessidade de poucas simulações;

- Estimativa de Configuração (Configuration Estimation): com a configuração dos parâmetros de configuração definidas e as restrições temporais definidas pelo projetista, são estimadas estaticamente, ou seja, sem utilização de simulações, quais os valores que os parâmetros do sistema (System Parameters) devem ter para apresentar o comportamento esperado. Esta etapa do fluxo também é automatizada e depende da viabilidade das definições realizadas pelo projetista, uma vez que o conjunto de informações coletadas nas simulações pode levar a uma configuração inválida.

Após a calibração do sistema com parâmetros que satisfazem às restrições do projeto, o modelo executável do sistema pode ser configurado para realização da simulação da plataforma. Esta etapa de simulação é tema da próxima subseção e permite que o projetista analise o espaço de projeto.

Nesta subseção foi fornecida uma visão geral de como a calibração do modelo é realizada, estando omitidos os detalhes sobre alguns conceitos da calibração, como o modelo de tempo ou parâmetros de sistema. Para que o leitor seja capaz de entender completamente esta etapa do fluxo de desenvolvimento é essencial que o capítulo 5 que trata das estimativas de tempo simulado seja cuidadosamente analisado.

### 4.2.3 Simulação da Plataforma

Com a modelagem dos componentes de sistema e sua calibração para atender os requisitos do projeto, são gerados a versão executável do modelo do sistema e os parâmetros de configuração do sistema para os modelos criados, respectivamente. Neste ponto do fluxo é feita a simulação nativa, ou seja, utilizando o próprio sistema de desenvolvimento, para avaliar o comportamento dinâmico da plataforma.

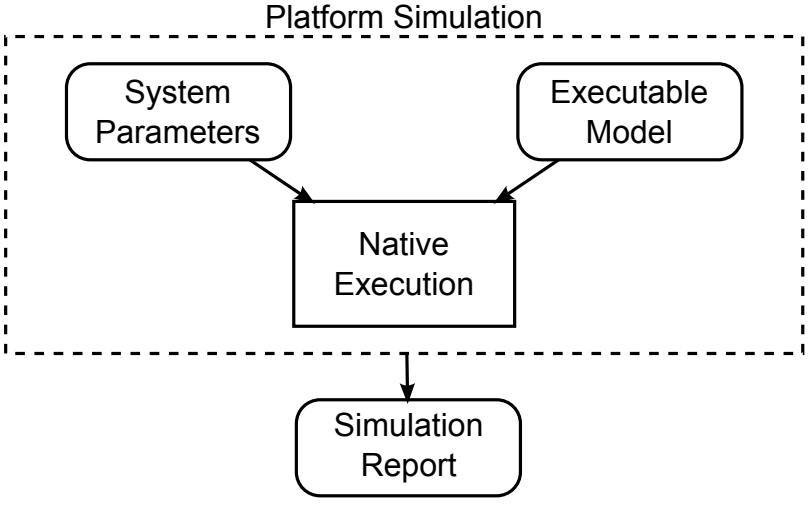


Figura 50: Fluxo de simulação da plataforma

O fluxo de simulação pode ser visualizado na figura 50, onde os parâmetros do sistema estimados na calibração (System Parameters) são passados para o modelo executável do sistema (Executable Model), para serem executados nativamente (Native Execution). O resultado desta execução é um relatório

de simulação (Simulation Report) que fornece ao projetista todas as informações referentes ao comportamento do sistema, como taxa de processamento ou tempo de resposta.

Estas informações sobre o comportamento dinâmico do sistema podem atender ou não as metas do projeto, podendo levar à modificações nas decisões de projeto realizadas na etapa de modelagem do sistema. É interessante observar que o sistema é modelado considerando os objetivos que precisam ser atingidos, sem que decisões tecnológicas, como, por exemplo, a escolha do processador, precisem ser tomadas prematuramente.

## 4.3 Mecanismos para Modelagem de Sistemas

Nesta seção serão detalhados os mecanismos de modelagem de sistemas proposto, utilizando uma estratégia de modelagem de cima para baixo (top-down), para exploração do espaço de projeto ou para o desenvolvimento de HdS na plataforma virtual.

### 4.3.1 Plataforma Virtual

A plataforma virtual é uma representação ou abstração de uma implementação real, utilizando camadas de software, como já foi visto no capítulo 2 de conceitos básicos, podendo estar definida em vários níveis e com propósitos diferentes. O objetivo desta subseção é definir uma plataforma base que será utilizada como referência em todas as subseções a seguir, definindo os elementos da aplicação, do processador e dos periféricos modelados.

O aspecto mais importante de ser observado aqui é que a plataforma é concebida para ter o mesmo comportamento, seja com o modelo de processador de alto nível proposto ou com um modelo ISS do mesmo, utilizando a mesma aplicação e sem necessidade de modificação no software da aplicação. Além desta compatibilidade de implementações, o modelo virtual de plataforma também permite que, em estágios iniciais do projeto, as estimativas geradas possam ser avaliadas, medindo sua precisão e a melhoria de desempenho oferecida.

Na figura 51, é ilustrada a plataforma virtual de referência usada como

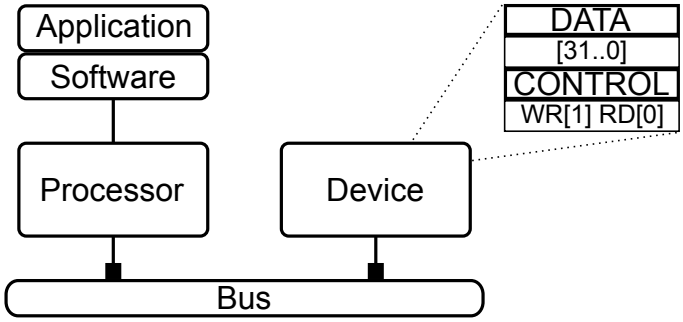


Figura 51: Plataforma virtual de referência

exemplo na modelagem dos componentes de hardware e software, sendo composta pela aplicação (Application) e seu módulo de software (Software); pelo módulo processador (Processor) com interface TLM; o barramento (Bus) para interconexão dos dispositivos, com informações de tempo; e um dispositivo hipotético (Device), que possui dois registradores de 32 bits para dados (DATA) e para controle (CONTROL), com campos de 1 bit de leitura (RD) e escrita (WR), nas posições 0 e 1, respectivamente.

Para exemplificar como a plataforma virtual é especificada, o código fonte necessário para sua modelagem é descrito no código fonte 4.1, onde todos os componentes são declarados, instanciados e conectados. Nas linhas 1 a 8, são feitas as inclusões de todos os cabeçalhos de todos os componentes necessários, como da biblioteca HdSC, dos modelos de sistema (processador e software) e os periféricos do barramento e o dispositivo hipotético.

Código Fonte 4.1: Código da plataforma virtual

```
1  // HdSC library include
2  #include <hdsc.hpp>
3  // HdSC system models include
4  #include <processor.hpp>
5  #include <software.hpp>
6  // Platform peripherals include
7  #include <bus.hpp>
8  #include <device.hpp>
9  /**
10  * Main function
11  */
12 int main(int argc, char* argv()) {
```

```
13    // Instantiating processor
14    processor Processor("Processor");
15    // Instantiating software
16    software Software(argv(3));
17    // Instantiating bus
18    bus Bus("Bus");
19    // Instantiating device
20    device Device("Device");
21    // Binding processor to bus
22    Processor.BUS(Bus);
23    // Binding timer to bus
24    Device.BUS(Bus);
25    // Setting processor CPI (0.5)
26    // and clock period (1 ns)
27    Processor.timing(0.5, 1);
28    // Executing software
29    Processor.execute(Software, argc, argv);
30    // Starting HdSC simulator
31    hsc_start();
32    // Returning success
33    return 0;
34 }
```

No escopo da função principal (linhas 9 a 34), todos os componentes são instanciados e conectados utilizando a interface escolhida com o barramento, além da execução do módulo de software pelo módulo de processador através da chamada de função execute, definida pelo projetista, passando o software e seus argumentos como parâmetros. Nas linhas 25 a 27, os parâmetros de tempo simulado do processador são ajustados pelo procedimento timing que é definido pela modelagem proposta e que recebe os valores de ciclos por instrução (CPI) e do período de relógio (clock period) com os quais o modelo do processador irá operar. Caso estes valores não sejam definidos, o modelo possui os valores de CPI e de período de relógio iguais a 0,5 e 1 nanosegundos definidos como padrão.

Nas linhas 28 a 31, o simulador é inicializado, fazendo com que todos os

componentes comecem a funcionar, realizando seu comportamento até que a condição de término da aplicação seja atingida. For fim, com o término da aplicação, caso não seja gerada nenhuma exceção, o processo retorna o código zero (linhas 32 e 33) para o sistema operacional, indicando que o processo terminou com sucesso e sem erros.

### 4.3.2 Aplicação

A aplicação é a implementação do comportamento do sistema, utilizando uma linguagem de programação de alto nível como C ou C++. Como já foi visto, na abordagem mais adotada de ISS, o código da aplicação é compilado e é gerado um código binário que é armazenado em uma memória. Este código binário possui uma sequência de instruções que são lidas, decodificadas e executadas pelo ISS ou pelo processador real. Desta forma, é observado que a granularidade de execução da aplicação é definida pela quantidade de operações atômicas, como blocos básicos ou instruções, que são realizadas em um determinado espaço de tempo.

A compilação do código fonte da aplicação da plataforma para ser executada no sistema nativo de desenvolvimento permite um maior desempenho em sua execução, em comparação com abordagens como o ISS que emulam o comportamento de instruções de uma determinada arquitetura de processador. Esta melhoria no desempenho é uma consequência da execução direta do código binário gerado no hardware do sistema de desenvolvimento, sem os custos de emulação de outra plataforma. A desvantagem desta abordagem de execução nativa está no isolamento da execução do software do ambiente de simulação da plataforma virtual, que é uma emulação de uma plataforma, impedindo que mecanismos de comunicação e interrupção relacionados aos componentes do sistema possam ser co-simulados. Para combinar o alto desempenho da execução nativa e os requisitos de integração com a plataforma virtual, é necessário definir qual será a granularidade da execução nativa do software e como será feita a interface de programação para a comunicação deste software com os dispositivos modelados na plataforma.

Para definição da granularidade de execução e especificação de interface de comunicação são propostas duas técnicas, respectivamente: instrumentação de código fonte em nível de bloco básico e uma interface de soft-

ware baseada em funções e registradores, que serão detalhadas nas subseções a seguir.

#### 4.3.2.1 Instrumentação de Código

Como acontece no ISS, que trabalha com granularidade de instruções, é preciso dividir a aplicação em unidades atômicas, que são os blocos básicos. Os blocos básicos são quaisquer sequências de operações que não sofrem desvios de fluxo durante sua execução. Então, cada bloco básico é uma unidade de processamento, permitindo a preempção entre eles, de forma análoga ao que o ISS faz com as instruções. Este processo de granularização dos blocos básicos é viabilizado com a instrumentação de código, que é um processo automático, realizado na etapa de preprocessamento do código da aplicação.

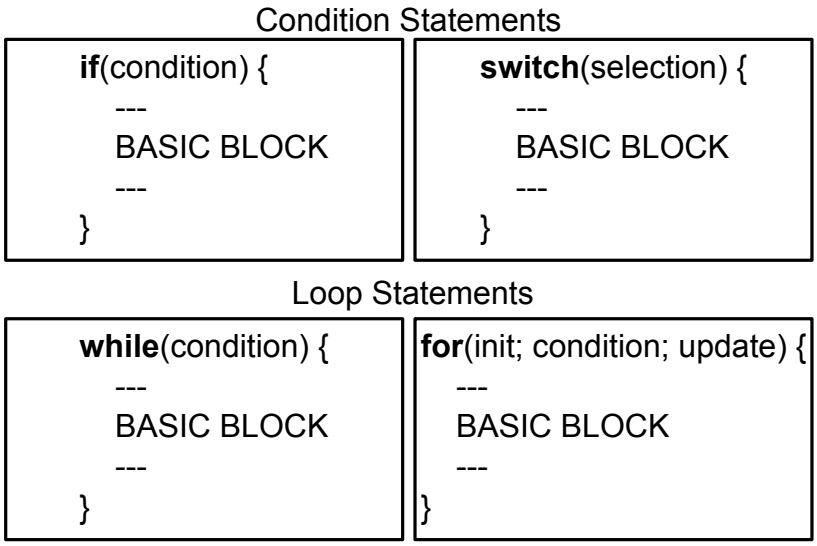


Figura 52: Definição de blocos básicos

Na figura 52, é possível observar como os blocos básicos são definidos e quais as sentenças de controle de fluxo que os definem. Na parte superior estão as sentenças condicionais se (if) e selecione (switch), que se baseando em uma condição ou valor, respectivamente, podem desviar o fluxo de execução da aplicação. De forma análoga, as sentenças de controle iterativo na parte inferior, enquanto (while) e para (for), também modificam o fluxo de execução, podendo repetir um bloco básico ou continuar no fluxo de execução.

Em cada um destes pontos de decisão, seja para checar a condição ou valor da seleção, é inserido, automaticamente pelo preprocessador do compilador através da inclusão do cabeçalho hsc_wrapper.hpp, uma função de checagem que permite a preempção do fluxo naquele ponto. Desta forma, sempre que estes controles de fluxo são utilizados, o código da aplicação é instrumentado para permitir que, em nível de bloco básico, seu fluxo possa ser interrompido pelas solicitações e resumido ao fim da execução das rotinas de tratamento das interrupções.

Existe um pequeno impacto da instrumentação de código no desempenho do sistema, devido a inclusão de comportamento adicional, mas este código adicional é fixo e só depende do número de blocos básicos da aplicação. Logo, a complexidade de tempo da aplicação é preservada, somente tendo um incremento uniforme nas constantes de cada bloco. Este tópico será mais bem detalhado no capítulo 5 que trata das estimativas de tempo simulado.

#### 4.3.2.2 Interface de Comunicação

É fundamental que, além de ser capaz de ter seu fluxo desviado, o código da aplicação possa incluir construtores ou funções para comunicação com os diversos componentes da plataforma. Para que isto possa ser feito foi desenvolvida uma interface de programação, que é baseada em funções e em registradores para acesso direto aos dispositivos. Com a possibilidade de acesso direto aos registradores, existe o suporte ao desenvolvimento nativo de HdS, que é uma parte crítica da aplicação.

No código fonte 4.2 são definidas as interfaces de comunicação através de registradores, com informações sobre o endereçamento do periférico na plataforma, seus nomes e campos disponíveis. Considerando um cabeçalho criado para múltiplas plataformas, podem ser utilizadas diretivas de preprocessamento (#ifdef e #else) que em tempo de compilação são capazes de escolher automaticamente qual trecho do código fonte deverá ser incluído.

Código Fonte 4.2: Exemplo de cabeçalho de plataforma em C

```
1 // Real hardware header
2 #ifdef __REAL_HW__
```

```
3  // Standard I/O include
4  #include <stdio.h>
5  // Register address definitions
6  #define DEVICE_DATA_ADDRESS (0x80000000)
7  #define DEVICE_CONTROL_ADDRESS (0x80000004)
8  // Pointer definition
9  unsigned int* PTR = DEVICE_DATA_ADDRESS;
10 // Register structure definitions
11 typedef struct CONTROLbits {
12   unsigned int    : 30;
13   unsigned int WR : 1;
14   unsigned int RD : 1;
15 }
16 CONTROLbits;
17 // Register aliases definitions
18 #define DATA (*((volatile unsigned int*)(DEVICE_DATA_ADDRESS)))
19 #define CONTROL (*((volatile unsigned int*)(
       DEVICE_CONTROL_ADDRESS)))
20 #define CONTROLbits (*((volatile CONTROLbits*)(
       DEVICE_CONTROL_ADDRESS)))
21
22 // HdSC models header
23 #else
24 // HdSC software include
25 #include <software.hpp>
26 // HdSC wrapping header for source code instrumentation
27 #include <hsc_wrapper.hpp>
28 #endif
```

Nas linhas 1 a 20 estão as definições do hardware real, seja ele implementado por um modelo ISS ou pelo próprio dispositivo físico. Todo o código fonte utilizado inclui cabeçalhos padrões (linhas 3 e 4), define o endereçamento dos registradores (linhas 5 a 7), define um ponteiro para aplicação acessar o registrador de dados do dispositivo (linhas 8 e 9), cria uma estrutura para os campos do registrador (linhas 10 a 16) e a própria declaração dos registradores (linhas 17 a 20).

Quando a compilação é feita nativamente e utilizando a modelagem de sistema proposta, o código fonte descrito nas linhas 22 a 28 é incluído. Todas as definições de gerenciamento de interrupção e de registradores, como endereçamento, nomes e campos, são definidos no módulo de software incluído nas linhas 24 e 25. Para realizar a instrumentação do código da aplicação, é necessária a inclusão do cabeçalho de instrumentação, como pode ser visto nas linhas 26 e 27.

É importante perceber que este cabeçalho não possui dependências da aplicação e somente especifica os componentes e informações da plataforma. Assim, este mesmo cabeçalho pode e deve ser utilizado em diversas aplicações que executem nesta plataforma, sem necessidade de modificações, desde que a plataforma se mantenha inalterada.

No cabeçalho da aplicação, visto no código fonte 4.3, são especificadas as definições da aplicação com a listagem das funções utilizadas pela aplicação. Nesta descrição são criadas suas seções para compilação para a plataforma alvo (hardware real) e para o modelo proposto que demanda o uso de construtores para o encapsulamento das funções no módulo de software que será descrito em detalhes na próxima seção.

Código Fonte 4.3: Exemplo de cabeçalho de aplicação em C

```
1  // Real hardware header
2  #ifdef __REAL_HW__
3  // Application function prototypes
4  void device_isr();
5  unsigned int device_read();
6  void device_write(unsigned int data);
7
8  // HdSC models header
9  #else
10 // Application function prototypes encapsulation
11 #define device_isr(...)      __ENCAP__(device_isr, __VA_ARGS__)
12 #define device_read(...)     __ENCAP__(device_read, __VA_ARGS__)
13 #define device_write(...)    __ENCAP__(device_write, __VA_ARGS__)
14 #endif
```

Nas linhas 3 a 6, as funções disponíveis na aplicação são listadas, permi-

tindo o seu uso mesmo fora do escopo de sua implementação, no âmbito da compilação para plataforma alvo. Para atingir este mesmo efeito no modelo de software, é preciso também listar estas funções e realizar seu encapsulamento ou incorporação ao modelo, como pode ser visto nas linhas 10 a 13. Utilizando estas definições de encapsulamento, os protótipos são declarados e todos os seus parâmetros são implicitamente definidos através de parâmetros de aridade (__VA_ARGS__) da linguagem de programação C.

Para exemplificar como o código da aplicação pode ser utilizado, no código fonte 4.4 é desenvolvida um pequeno software para controle e acesso de um dispositivo hipotético, descrito em C. Como é um requisito utilizar o mesmo código da implementação no desenvolvimento e nas simulações, é feita a inclusão nas linhas 1 e 2 de um cabeçalho de plataforma que contém definições da plataforma real e da plataforma baseada em modelos de alto nível do sistema.

Código Fonte 4.4: Exemplo de código fonte de aplicação em C

```
1  // Platform definitions header
2  #include <platform.h>
3  // Application definitions header
4  #include <application.h>
5
6  // Counter variable
7  unsigned int counter = 0;
8
9  // Device ISR function
10 void device_isr() {
11   // Clearing control register
12   CONTROL = 0;
13 }
14
15 // Device read function
16 unsigned int device_read() {
17   // Data read request
18   CONTROLbits.RD = 1;
19   // Returning data incremented
```

```
20    return DATA + 1;
21  }
22
23  // Device write function
24  void device_write(unsigned int data) {
25    // Writing data register via pointer
26    *PTR = data;
27    // Data write request
28    CONTROLbits.WR = 1;
29  }
30
31  // Application main function
32  int main(int argc, char* argv()) {
33    // Loops for 256 iterations
34    while(counter < 256) {
35      // Device write
36      device_write(counter);
37      // Device read
38      counter = device_read();
39    }
40    // Returning success
41    return 0;
42  }
```

O comportamento da aplicação propriamente dita é descrita nas linhas 6 a 42, onde uma variável de contagem inteira e sem sinal é criada (linhas 6 e 7) e são implementadas funções para tratamento das interrupções geradas (device_isr) pelo dispositivo nas linhas 9 a 13, para leitura do dispositivo (device_read) nas linhas 15 a 21, uma função para escrita no dispositivo (device_write) nas linhas 23 a 29 e a função principal (main) nas linhas 31 a 42.

Toda vez que na função principal for feita uma requisição ao dispositivo hipotético, uma interrupção será gerada e uma rotina de tratamento de interrupção ou ISR será executada. Isto significa que ao término de cada requisição feita ao dispositivo será gerada uma interrupção para limpar o registrador de controle (linhas 11 e 12). No laço de 256 interações, o contador definido

globalmente é escrito no dispositivo através da função de escrita de acesso direto aos registradores (linhas 25 a 28) do dispositivo. Após a escrita é feita a leitura do dado escrito e seu incremento através da função de leitura do dispositivo, acessando os registradores de controle e de dados (linhas 17 a 20).

No modelo de aplicação apresentado fica evidente o suporte para a execução do software em nível de linguagem de programação C ou C++, permitindo preempção do seu fluxo de execução em nível de bloco básico e a comunicação com os dispositivos externos da plataforma. Tudo isto é feito usando o mesmo código da aplicação que será usado na implementação real, ou seja, não são criadas abstrações de comportamento, mas sim abstrações no processamento do sistema que impactam minimamente no código fonte da aplicação desenvolvida com o desenvolvimento de cabeçalhos de plataforma do modelo.

### 4.3.3 Módulo de Software

Como um recipiente para suportar a aplicação, o módulo de software incorpora todas as suas definições de dados e de código. Com um funcionamento análogo a um dispositivo de memória, que armazena o código binário da aplicação, o módulo de software define um componente que incorporará todas as funções e variáveis da aplicação, permitindo sua instanciação e conexão com o módulo de processador.

Para implementar a estrutura e comportamento do módulo de software, são propostas uma série de definições de linguagem que são descritas a seguir:

- HSC_SOFTWARE_TEMPLATE(address_type, data_type): esta diretiva tem o papel de definir, usando as técnicas de programação genérica, qual o tipo de dado que será usado para endereçamento (address_type) dos dispositivos e para acesso de seus registradores (data_type). Desta forma, é possível para uma mesma implementação do componente de software, lidar com diferentes tipos e tamanhos de endereçamento e dados da plataforma;
- HSC_SOFTWARE(name): define o identificador do componente (name) de software, que será utilizado para declaração e instanciação do com-

ponente. Esta definição está atrelada ao tipo de dado usado para endereçamento e dados, definido pela diretiva anterior, assim como ocorre na definição dos templates e classes de C++;

- HSC_SOFTWARE_CTOR(name): funcionando como construtor do componente de software, esta declaração permite inicializar todas as variáveis do módulo, através de atribuições diretas ou chamadas de rotinas de inicialização, como a função bind que define o endereçamento dos registradores declarados na plataforma;

- HSC_SOFTWARE_REGISTER(name): com esta diretiva, é permitido ao projetista definir uma especialização dos registradores dos dispositivos, definindo campos de acesso e seu mapeamento, além de definir seu identificador (name). Desta forma, são obtidas as mesmas capacidades que a aplicação possui quando utiliza os recursos de endereçamento de memória disponível no ambiente alvo de execução;

  - HSC_SOFTWARE_REGISTER_FIELD(field_name): dentro do escopo da declaração do registrador, esta diretiva é usada para definir os nomes dos campos (field_name) do registrador declarado no nível superior. Todos os campos são declarados sequencialmente usando esta diretiva, sem importar a ordem de sua declaração, pois seu posicionamento no registrador é definido pelo mapeamento;
  - HSC_SOFTWARE_REGISTER_MAP(field_list): quando todos os campos do registrador são definidos, é feito o mapeamento desta lista de campos (field_list), para que seu posicionamento e tamanho dentro do registrador sejam determinados. Assim, quando cada campo for acessado, seja para escrita ou leitura, uma máscara do valor do registrador será aplicado para que exatamente o número de bits definido seja acessado;
  - HSC_SOFTWARE_REGISTER_RANGE(field_name, start, length): no escopo do mapeamento, cada campo do registrador (field_name) tem sua posição inicial em bits definida (start) e seu tamanho (length) a partir daquela posição. Se o registrador em questão possui 32 bits de largura, existem índices de 0 até 31 para cada bit deste registrador.

- HSC_SOFTWARE_REGISTER_DECLARATION(name): esta diretiva cria um registrador com o identificador passado como parâmetro (name), sendo do mesmo tipo de dados definido na declaração do componente de software. Desta forma, a aplicação acessa este registrador com todas as operações disponíveis para o tipo especificado;

- HSC_SOFTWARE_REGISTER_CUSTOM_DECLARATION(type, name): com a capacidade de especialização dos registradores, é através desta diretiva que é feita a sua declaração, a partir do tipo definido (type) e do nome da variável (name) que estará visível para a aplicação. É importante notar que apesar de serem duas variáveis isoladas, o registrador e sua especialização são interligadas pela diretiva bind, vista no construtor do componente;

- HSC_SOFTWARE_POINTER_DECLARATION(name): esta diretiva cria um ponteiro com o identificador passado como parâmetro (name), através do qual é possível realizar acesso direto de endereços da plataforma virtual. Logo, a aplicação é capaz de fazer acesso direto aos componentes da plataforma, apesar de executar nativamente e utilizar a sintaxe padrão de ponteiros.

Para incorporação das funções e variáveis globais da aplicação, são definidas para módulo de software mais três diretivas de programação:

- HSC_SOFTWARE_ENCAPSULATION(return_type, function, args): diretiva com a função de tornar a função da aplicação um membro do componente de software, permitindo seu acesso protegido e encapsulado, inclusive isolando as pilhas de execução concorrentes. Recebe como parâmetros o retorno da função da aplicação (return_type), o seu nome (function) e a seguir todos os seus parâmetros (args) separados por vírgula;

- HSC_SOFTWARE_GLOBAL(data_type, variable_name): com função análoga ao encapsulamento das funções, esta diretiva encapsula as variáveis globais da aplicação, gerando proteção para acesso de seu conteúdo e possibilitando que múltiplas instâncias possam ser simuladas de forma independente. Sua sintaxe possui dois parâmetros, o primeiro é o tipo da variável (data_type) e o segundo é o nome da variável (variable_name), sendo separados por vírgula;

- HSC_SOFTWARE_MAIN(name): diretiva com comportamento similar a de encapsulamento de funções, tendo o propósito de definir o identificador da função principal (name) da aplicação, permitindo um ambiente de múltiplas aplicações. Desta forma, no módulo de software, pode ser feita a checagem de que programa está sendo invocado e é feita a sua execução chamando pelo identificador que foi definido.

Um exemplo de implementação do módulo de software é fornecido no código fonte 4.5, com foco nos construtores propostos e omitindo todas as demais funcionalidades de livre implementação, ou seja, que estão fora do escopo da linguagem proposta e que podem ser definidos como o projetista achar mais conveniente.

Código Fonte 4.5: Exemplo de implementação do módulo de software

```
1  /**
2   * Software Module declaration
3   */
4  HSC_SOFTWARE_TEMPLATE(address, data);
5  HSC_SOFTWARE(software) {
6    // Software module constructor
7    HSC_SOFTWARE_CTOR(software) {
8      // Binding pointer
9      bind(PTR);
10     // Binding registers address
11     bind(DATA, 0x80000000);
12     bind(CONTROL, CONTROLbits, 0x80000004);
13     // Setting data pointer address
14     PTR = 0x80000000;
15     // Global variable initialization
16     counter = 0;
17   }
18
19   // Application main loader
20   void hsc_software_main() {
21     // Calling application main
22     main_application(hsc_argc(), hsc_argv());
```

```
23  }
24  // Application ISR handler
25  void hsc_software_interruption(const hsc_device_code& device)
        {
26    // Calling application ISR
27    device_isr();
28  }
29
30  // Software embedding directives for application source code
31  HSC_SOFTWARE_GLOBAL(unsigned int, counter);
32  HSC_SOFTWARE_ENCAPSULATION(void, device_isr, void);
33  HSC_SOFTWARE_ENCAPSULATION(unsigned int, device_read, void);
34  HSC_SOFTWARE_ENCAPSULATION(void, device_write, unsigned int
      data);
35  HSC_SOFTWARE_MAIN(application);
36
37  // DATA pointer
38  HSC_SOFTWARE_POINTER_DECLARATION(PTR);
39
40  // CONTROL bits definition
41  HSC_SOFTWARE_REGISTER(CONTROLbits) {
42    // CONTROL fields
43    HSC_SOFTWARE_REGISTER_FIELD(WR);
44    HSC_SOFTWARE_REGISTER_FIELD(RD);
45    // CONTROL mapping
46    HSC_SOFTWARE_REGISTER_MAP
47    (
48      // Setting ranges
49      HSC_SOFTWARE_REGISTER_RANGE(WR, 1, 1);
50      HSC_SOFTWARE_REGISTER_RANGE(RD, 0, 1);
51    )
52  };
53  // DATA register
54  HSC_SOFTWARE_REGISTER_DECLARATION(DATA);
55  // CONTROL register
```

```
56    HSC_SOFTWARE_REGISTER_DECLARATION(CONTROL);
57    // CONTROL bits register
58    HSC_SOFTWARE_REGISTER_CUSTOM_DECLARATION(CONTROLbits,
          CONTROLbits);
59  };
```

A implementação tem início com a declaração de tipos de dados que serão utilizados para o endereçamento (address) e para acesso de dados (data) na plataforma, através da aplicação de programação genérica com templates (linha 4). Uma vez definidos os tipos de dados, é preciso declarar o módulo de software (linha 5) e informar qual o seu nome ou identificador que será utilizado como tipo durante a instanciação do componente. Para inicializar as declarações realizadas é definido um construtor para o módulo (linhas 6 a 17). O ponteiro especificado para aplicação é conectado ao módulo de software (linhas 8 e 9), os endereços dos registradores modelados são mapeados (linhas 10 a 12) e as variáveis globais da aplicação são inicializadas (linhas 13 e 16).

As interfaces para execução da aplicação (hsc_software_main) e de tratamento das interrupções (hsc_software_interruption) estão definidas no modelo de software e precisam ser implementadas. Na implementação da interface de execução do software (linhas 19 a 23), é feita a chamada da função principal da aplicação e o repasse dos argumentos de execução. Por ser um exemplo, só existe a chamada para uma função, entretanto, o projetista pode utilizar este mesmo módulo de software para invocar várias aplicações diferentes, bastando que seja implementado um mecanismo de seleção.

Para o tratamento das interrupções, é feita a implementação da interface de interrupção (linhas 24 a 28) que é executada toda vez que uma interrupção é gerada na plataforma e repassando o código do dispositivo que fez a requisição. Mais uma vez, a rotina de tratamento da aplicação (device_isr) é chamada diretamente, mas é possível modelar um sistema programável de gerenciamento de interrupções que pode ser utilizado por várias aplicações distintas, necessitando do projetista sua implementação e concepção.

Seguindo as linhas de implementação, é feito o encapsulamento das variáveis e funções da aplicação (linhas 30 a 35) para permitir o seu uso dentro no módulo de software, permitindo múltiplas instâncias com proteção das informações. Este suporte é fundamental para execução em plataformas mul-

tiprocessadas onde podem existir múltiplas instâncias da aplicação na plataforma que devem funcionar de forma isolada e independente, apesar de serem compiladas do mesmo código fonte.

A parte mais importante da modelagem está na definição da interface baseada em ponteiro (linhas 37 e 38) e em registradores (linhas 40 a 58), onde os registradores e seus campos são definidos, utilizando os construtores propostos. Quando se deseja definir os campos de um registrador, é preciso utilizar o construtor de registrador e informar o identificador do registrador com campos (linhas 40 e 41) para que possam ser definidos os campos (linhas 42 a 44) e seu mapeamento absoluto dentro do registrador (linhas 45 a 51). Caso somente se deseje declarar um registrador sem seus campos, basta que seja feita sua declaração (linhas 53 a 56), sem maiores detalhes. Por fim, é preciso declarar o tipo de registrador com campos modelados (linhas 57 e 58), para que possa ser combinado ao registrador principal no construtor do módulo (linha 12).

É importante deixar claro que este componente é reusável e deve ser utilizado para suportar múltiplas aplicações, sem que sejam necessárias muitas modificações, como já foi dito anteriormente. Este aspecto é essencial para otimizar o processo de desenvolvimento, causando o mínimo de impacto na implementação do sistema.

### 4.3.4 Módulo de Processador

Este módulo é uma abstração da unidade de processamento, fornecendo para a aplicação o acesso aos recursos da plataforma e todos os mecanismos inerentes do hardware, como acesso ao barramento de interconexão e controle de interrupção. Apesar de ser uma abstração de modelos precisos, como o ISS, este módulo tem como pré-requisito a existência do módulo de software, para armazenar a aplicação que será carregada e executada pelo processador.

Desta maneira, o hardware e o software podem ser simulados exatamente nas mesmas linguagens em que são implementados. Assim, o software executando sobre o hardware virtual encontra exatamente as mesmas interfaces e funções do modelo baseado em ISS do processador ou hardware real, mas abstraindo uma série de detalhes e, com isto, reduzindo o seu tempo de de-

senvolvimento e melhorando dramaticamente o desempenho da simulação.

Como os demais componentes propostos, uma série de diretivas de programação foi desenvolvida para implementar o módulo de processador, que são:

- HSC_PROCESSOR_TEMPLATE(address_type, data_type): define a parametrização de programação genérica, permitindo que o modelo criado trabalhe com diferentes tipos de dados para endereçamento (address_type) dos dispositivos e acesso de suas informações (data_type). Com isto, diferentes níveis de transação podem ser utilizados, com diferentes tipos de dados e níveis de granularidades. A única restrição que se aplica é que ambos os tipos de dados devem ser iguais ao módulo de software, para que sejam compatíveis e permitam a troca de informações;
- HSC_PROCESSOR(name): esta diretiva cria o módulo de processador propriamente dito, usando os tipos de endereço e dados definidos pelo template, gerando um componente de implementação com o identificador (name) passado como parâmetro. Este identificador criado é usado para declarar e instanciar este componente na construção da plataforma virtual de simulação, podendo ter múltiplas instâncias para multiprocessamento;
- HSC_PROCESSOR_CTOR(name): é o construtor do módulo de processador, recebendo como parâmetro o mesmo identificador do módulo (name). Serve como ponto de início do modelo de processamento, ajustando as variáveis de configuração através de rotinas de inicialização ou iniciando os processos de controle necessários.

Para exemplificar como um módulo de processador é implementado, é fornecido o código fonte 4.6 que ilustra a utilização dos construtores da linguagem proposta. A modelagem tem início com a definição dos tipos de dados que serão utilizados para endereçamento (address) e para acesso aos dados (data). Para isto, é empregada a diretiva de programação genérica baseada em templates (linha 4) que define os tipos de dados e devem ser compatíveis com os utilizados no modelo de software.

Código Fonte 4.6: Exemplo de implementação do módulo de processador

```
1  /**
2   * Processor Module declaration
3   */
4  HSC_PROCESSOR_TEMPLATE(address, data);
5  HSC_PROCESSOR(processor) {
6    // Processor constructor
7    HSC_PROCESSOR_CTOR(processor) {
8      // Starting up interruption handler
9      SC_THREAD(interruption_handler);
10   }
11
12   // TLM Bus port
13   bus_port BUS;
14
15   // SystemC Interruption handler process
16   void interruption_handler() {
17     // Loop forever
18     while(1) {
19       // Waiting interruption event
20       wait(BUS->get_interrupt_event());
21       // Retrieving device code
22       hsc_device_code_t device_code = BUS->
             get_interrupt_device_code();
23       // Calling my software interruption handler
24       hsc_processor_interruption(device_code);
25       // Bus interruption acknowledgment
26       BUS->interruption_acknowledge();
27     }
28   }
29
30   // Processor read call back implementation
31   void hsc_processor_read(const address& address, data& data) {
32     // Calling bus read
33     BUS->read(address, data);
34   }
```

```
35    // Processor write call back implementation
36    void hsc_processor_write(const address& address, const data&
          data) {
37      // Calling bus write
38      BUS->write(address, data);
39    }
40  };
```

Na declaração do módulo de processador (linha 5), é feita a definição do seu identificador que será utilizado para declaração e instanciação do componente na plataforma virtual. No construtor do módulo (linhas 6 a 10) é feita a inicialização de um processo para tratar os eventos referentes às interrupções geradas pela plataforma. Estas interrupções são recebidas pela interface de comunicação com o barramento (linhas 12 e 13), que pode ser implementada em qualquer outro padrão de interface ou nível de abstração que se deseje.

O processo de gerenciamento de interrupção do processador (linhas 15 a 28) é dependente do tipo de interface de comunicação escolhida, podendo o projetista implementar este processo como achar mais conveniente ou até mesmo dispensar sua implementação com o uso de outras abordagens, como entrada e saída programada. Neste exemplo, o processo executa infinitamente (linhas 17 e 18) e tem seu funcionamento iniciado com a geração de um evento de interrupção (linhas 19 e 20). Quando o evento é gerado, é feita a captura do código do dispositivo que fez a solicitação (linhas 21 e 22) e é feita a chamada da rotina de tratamento do módulo de software (linhas 23 e 24). Com o término da execução da rotina de software, a requisição é finalizada com o reconhecimento da solicitação no barramento (linhas 25 e 26).

Para redirecionar as requisições de acesso à plataforma feitas pelo módulo de software, é obrigatória a implementação das interfaces de leitura (hsc_processor_read) e de escrita (hsc_processor_write) do módulo de processador (linhas 30 a 39). Nestas implementações as requisições de leitura (linhas 30 a 34) e de escrita (linhas 35 a 39) são traduzidas para a interface de barramento escolhida. Mais uma vez, esta funcionalidade pode ser feita de várias maneiras e o objetivo deste exemplo é ilustrar como funciona este processo.

Assim como o módulo de software, o módulo de processador é reusável

e pode ser utilizado para executar diferentes aplicações, podendo estar contido em diferentes plataformas e possuindo múltiplas instâncias para suportar multiprocessamento.

## 4.4 Técnica Proposta para Simulação de Sistemas

Para suportar os diferentes níveis de abstração vistos e os mecanismos de modelagem propostos, foi desenvolvido um conjunto de componentes que estão ilustrados na figura 53. O objetivo principal desta seção é detalhar como a modelagem proposta é suportada, descrevendo como foi organizada a sua implementação e quais foram os requisitos considerados. Neste primeiro momento, os construtores de modelagem são enumerados e os requisitos elicitados que foram implementados estão descritos:

- Aplicação (Application): consiste no mesmo código fonte em linguagem de programação C ou C++ que será compilado para a plataforma alvo. Seguindo o mesmo ponto chave do simulador de conjunto de instrução, o projetista tem a certeza que ele está desenvolvendo e simulando a mesma descrição que será utilizada para implementação, com a diferença de estar fazendo isto em modelos de processamento, com diferentes níveis de detalhe e desempenho;

- Módulo de Software (Software Module): proporciona ao código da aplicação todas as interfaces necessárias para seu encapsulamento, abstraindo o conceito de armazenamento em memória, e acesso aos dispositivos da plataforma, através de uma interface de programação de alto nível e flexibilidade;

  - Encapsulamento: o módulo de software define uma classe para encapsulamento de todas as funções e variáveis da aplicação, gerando um modelo que abstrai o papel de armazenamento da memória binária da aplicação compilada. Desta forma, com este encapsulamento, é possível ter múltiplas instâncias desta memória de programa, composta pela aplicação e módulo de software, que tenham o mesmo isolamento e comportamento observado nas memórias que armazenam o código binário do programa;

- Acesso aos dispositivos: utilizando uma interface de programação para a aplicação ou Application Programming Interface (API), o módulo de software disponibiliza para a aplicação todas as definições e assinaturas necessárias para acessar as funções da plataforma. Estas funções são referentes ao acesso de dispositivos, pela escrita e leitura de seus registradores, gerenciamento de interrupções através da implementação de rotinas de tratamento em software e esquema de carregamento e inicialização da aplicação. É papel do desenvolvedor conceber e implementar as interfaces deste componente, permitindo que o software da aplicação tenha seus requisitos de acesso à plataforma atendidos.

- Módulo de Processador (Processor Module): seguindo no paradigma de abstração de processamento, este componente é responsável por modelar o comportamento fornecido pelo ISS. Entretanto, ao invés de simular o comportamento em nível de código binário e instruções contidas em uma memória, o próprio código fonte da aplicação em linguagem de programação de alto nível é simulado. Este código da aplicação é representado pelos componentes de aplicação e software integrados, oferecendo uma abstração estrutural e comportamental do que acontece na abordagem ISS.

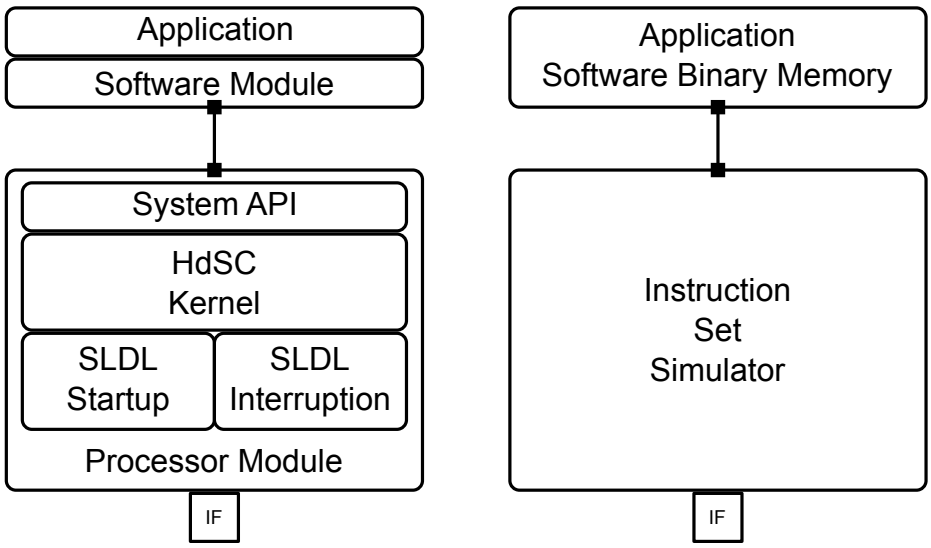


Figura 53: Arquitetura HdSC (esquerda) versus Arquitetura ISS (direita)

O elemento principal desta organização, que abstrai o modelo de processamento, é o módulo de processador (Processor Module), que de fato é

quem executa todas as funções de inicialização, execução e escalonamento da aplicação, além das interfaces de acesso para hardware e software. Para contemplar estas funcionalidades, foram desenvolvidos diversos subcomponentes (vistos na figura 53), para suportar este modelo, que são:

- Interface de Sistema (System API): foi utilizada uma interface padronizada, como a interface POSIX (60), para executar o software da aplicação da plataforma de forma nativa, com todos os recursos de programação e desenvolvimento disponíveis. Isto implica em usar todas as funcionalidades disponíveis pelo sistema operacional nativo, como programação concorrente e mecanismos de sincronização, de forma transparente e fornecendo para aplicação a visão de programação da plataforma modelada. Como grande parte dos sistemas operacionais, inclusive os embarcados, é aderente ao padrão POSIX, uma aplicação que seja compatível com esta interface pode ser facilmente integrada;
- Núcleo HdSC (HdSC Kernel): realiza todas as atividades de sincronização e comunicação entre o sistema nativo, que executa o fluxo do software da aplicação, e o sistema que simule a plataforma, que contém os modelos de tempo, de estruturas e de comportamentos dos dispositivos da plataforma. Além destas funções de sincronismo e escalonamento dos processos, o núcleo de simulação HdSC é responsável pelas estimativas de tempo e preempção do software da aplicação;
  - Comunicação: as requisições para comunicação externa da aplicação precisam ser adequadamente tratadas, de forma a permitir que exista consistência e coerência no processo. Para tanto, o núcleo de simulação é responsável por escalonar o processo de software e o processo da plataforma, evitando condições de corrida e permitindo que o software tenha um acesso eficiente aos recursos da plataforma. Este é um ponto crucial, para se ter o desenvolvimento de um sistema, onde o software é capaz de se comunicar com os dispositivos, utilizando as interfaces disponibilizadas;
  - Estimativa de tempo: como a aplicação é executada nativamente, não existe nenhuma relação entre o tempo de execução nativo e o tempo simulado na plataforma. É imprescindível que uma aplicação que executa nativamente possua esta noção de tempo simulado,

pois sem esta informação não é possível situar as operações e resultados gerados pela aplicação no tempo simulado da plataforma. Isto quer dizer que se uma aplicação não possui a noção de tempo, todas as suas operações começam e terminam em um tempo zero ou instantâneo;

- Preempção da aplicação: outro aspecto importante é a preempção da execução da aplicação, visando interromper seu fluxo para execução de fluxos alternativos ou de exceção para tratamento de interrupções geradas. Mais uma vez, por executar nativamente, esta função não é suportada pelo simulador da plataforma, assim como o tempo simulado, precisando que o núcleo de simulação realize todo o processo de interrupção, desvio e retorno, de forma transparente e consistente para o desenvolvedor do sistema.

- Suporte de Inicialização (SLDL Startup): este componente é responsável pelas sequências de inicialização da aplicação, preparando os segmentos de código e definindo parâmetros, como a execução da função principal (main). Estes segmentos de código incluem a configuração de estruturas para alocação dinâmica de memória, como pilha e heap, além das demais variáveis declaradas pela aplicação. Todos estes aspectos dependem diretamente no nível abstração escolhido, uma vez que diferentes níveis de detalhamento são necessários para a execução dos componentes, e sua implementação é feita pelo módulo de software;

- Gerenciamento de Interrupção (SLDL Interruption): definindo a visão de hardware para o tratamento de interrupções, este componente recebe todas as possíveis fontes de interrupção que podem ser geradas da plataforma, criando as políticas para escalonamento e priorização das requisições. Uma vez que o evento é detectado, o núcleo HdSC recebe a solicitação para interromper a execução e desviar o fluxo de execução para rotina de tratamento da interrupção. Os aspectos referentes ao processador, como seus registradores internos e mecanismos de interrupção podem ser refletidos neste componente, permitindo um nível de detalhe tão grande quanto se deseje.

Para detalhes mais formais sobre a linguagem proposta, é fornecida uma gramática no formato Backus-Naur (BNF) (61) da linguagem proposta no anexo

A. Todos os construtores, suas sintaxes e parâmetros serão definidos nas seções a seguir, então o leitor deve considerar este anexo como uma referência mais formal e complementar da descrição deste capítulo.

### 4.4.1 Núcleo de Simulação

Para suportar a execução da modelagem proposta foi desenvolvido um núcleo de simulação que suporta a comunicação, escalonamento e sincronização entre o ambiente de plataforma virtual e a execução nativa do software. O comportamento do núcleo de simulação é ilustrado na figura 54, contendo informações sobre os seus estados e suas transições. Os processos do núcleo de simulação podem ser resumidos em: operações de sincronismo e escalonamento entre a execução do software e da simulação da plataforma; gerenciamento de interrupção no software sendo ativado por dispositivos de hardware da plataforma; e requisições para operações de E/S para permitir ao software o acesso de dados dos dispositivos de hardware.

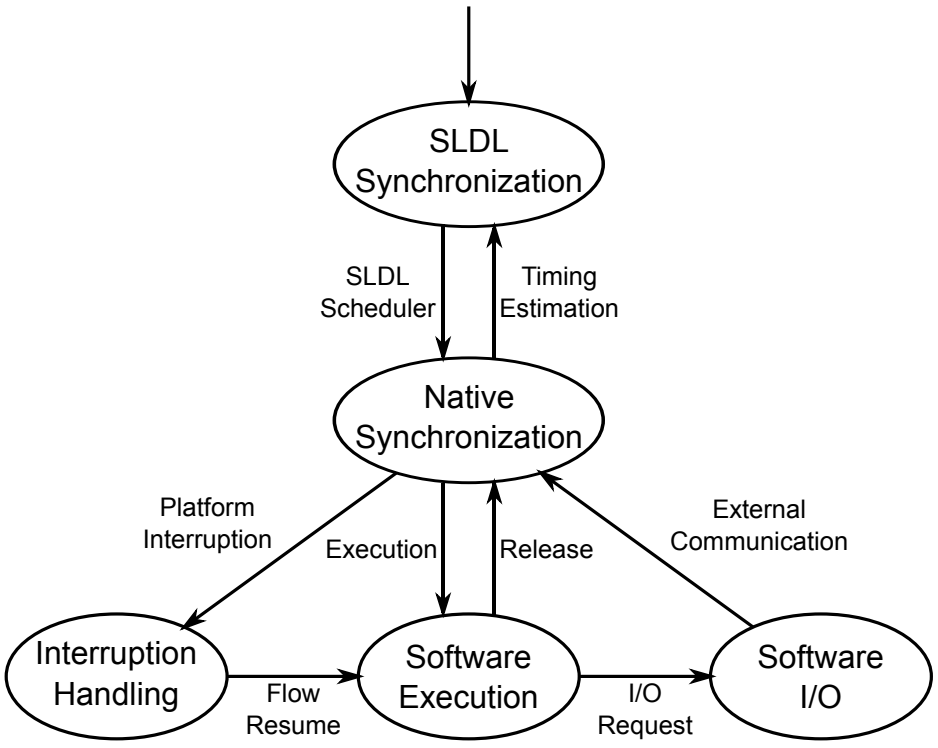


Figura 54: Máquina de estados do núcleo de simulação

- Sincronização da Plataforma (SLDL Synchronization): com o início da simulação, todos os processos de inicialização do núcleo da SLDL, que

neste caso é SystemC, são executados, permitindo a instanciação dos componentes da plataforma, elaboração das interfaces e suas conexões. Neste estado do núcleo de simulação, que pode ser visto na figura 54, seu comportamento entra em suspensão até que o ponto de sincronismo seja atingido e as operações pendentes sejam realizadas;

  - Escalonador SLDL (SLDL Scheduler): esta transição é realizada quando o núcleo de simulação da plataforma virtual escalona a execução nativa do software, realizando as operações de sincronismo necessárias para garantir o funcionamento consistente do sistema. As políticas de escalonamento da SLDL e das tarefas do sistema nativo não são definidas ou modificadas pelo escalonador proposto, uma vez seu propósito é permitir a integração entre os componentes da plataforma virtual e o software que executa nativamente;
  - Estimativa de tempo (Timing Estimation): uma vez que a execução nativa de parte do fluxo do software é finalizada, é feita a estimativa de atraso equivalente do tempo de execução na plataforma virtual, repassando o controle de simulação para o núcleo da SLDL.

- Sincronização do Núcleo HdSC (Native Synchronization): uma vez que a simulação da plataforma está em andamento, o escalonador proposto realiza operações de sincronização para executar nativamente o software da aplicação. Estas operações de sincronização causam uma exclusão mútua entre a execução da plataforma virtual e a execução nativa, evitando conflitos decorrentes de concorrência. Nesta etapa, ilustrada na figura 54, além das funções de sincronização, o núcleo de simulação realiza as estimativas de tempo simulado referente ao trecho de software executado no ambiente de simulação da plataforma virtual. A transição para outros estados pode acontecer de acordo com os eventos descritos a seguir:

  - Interrupção na plataforma (Platform Interruption): é ativada quando um evento de interrupção é gerado na plataforma e o processador precisa interromper o fluxo de execução do software, retomando a sua execução (Flow Resume) após finalização da rotina de tratamento;

  - Execução e liberação do software (Execution / Release): acontece quando a plataforma sincroniza e o núcleo de simulação proposto realiza a execução nativa do software, gerenciando a liberação dos recursos para que outras operações sejam realizadas, como tratamento de interrupção e operações de E/S (I/O Request);
  - Comunicação externa (External Communication): a requisição de E/S durante a execução do software demanda uma sincronização com a plataforma para que o fluxo de dados entre a plataforma e o software executando nativamente seja realizado.

- Gerenciamento de Interrupção (Interruption Handling): quando a plataforma está sendo simulada e uma interrupção é gerada por algum dispositivo, é preciso interromper a execução do software. O escalonador de HdSC realiza a sincronização com a plataforma e faz a chamada da rotina de tratamento de interrupção no software da aplicação. O fluxo principal da aplicação é suspenso enquanto a rotina de tratamento está executando, tendo seu fluxo principal de execução restaurado após a finalização da rotina de tratamento de interrupção;

- E/S de Software (Software I/O): caso durante a execução do software seja feita uma requisição para realização de operação de entrada e saída (E/S), o núcleo de simulação realiza a suspensão do fluxo de execução do software e faz a sincronização com a plataforma para geração da transação no barramento. Após a finalização da operação de E/S solicitada, o fluxo de execução do software é restaurado e a aplicação continua sua execução até seu controle seja retornado pelo escalonador, solicite outra operação de E/S ou seja interrompido por algum dispositivo da plataforma;

- Execução de Software (Software Execution): se não existem pendências de interrupção ou de entrada e saída, o escalonador em sincronismo com a plataforma virtual realiza a execução nativa do software em intervalos de tempo parametrizados. Ao final de cada execução nativa de parte do software, durante a liberação para execução da plataforma, o tempo simulado referente a execução nativa é estimado, criando no ambiente virtual de simulação o atraso correspondente pela execução nativa do software.

Nas próximas seções vários detalhes da implementação do núcleo de simulação serão descritos, inclusive os referentes a plataformas com múltiplos processadores. Entretanto, nesta seção já se torna pertinente explicar como o núcleo de simulação suporta múltiplas instâncias de software na plataforma decorrentes de multiprocessamento. Em seu projeto, o núcleo de simulação possui um controle centralizado do escalonador HdSC, para todas as instâncias criadas, como forma de controlar quantas instâncias estão executando e possibilitar um sincronismo com o ambiente de plataforma virtual.

Este controle centralizado não impõe limites quanto ao número de instâncias que podem ser criadas e nem gera gargalos durante a realização das simulações, devido ao fato de cada instância executar de forma paralela na máquina nativa. As limitações de desempenho e escalabilidade desta organização são decorrentes da sincronização que se faz necessária com o núcleo de simulação sequencial de SystemC. Caso a implementação de SystemC fosse paralelizada, as operações de escalonamento e de sincronização não precisariam esperar a simulação sequencial de todos os processos da plataforma virtual, permitindo otimizar a utilização dos recursos de processamento do sistema nativo de desenvolvimento.

## 4.5 Suporte para Multiprocessamento

Com a crescente complexidade dos sistemas, a extração do paralelismo em nível de instrução ou Instruction Level Parallelism (ILP) (41) atingiu o seu limite. O paralelismo em nível de thread passou a ser explorado na maioria dos sistemas, desta forma é essencial que a abordagem proposta apresente um suporte para execução multiprocessada do software (42) para aumentar seu desempenho de execução. Caso o leitor deseje revisar os diferentes tipos de multiprocessamento, as suas características e os seus desafios de implementação é recomendada a leitura do capítulo 2 de conceitos básicos.

Por ter um foco na simulação do sistema em um nível mais alto de abstração através da execução nativa do software, a modelagem de cache de dados e instrução da aplicação não é suportada. A principal justificativa para esta falta de suporte está no fato do software ser executado nativamente, sem noção de instruções sendo executadas ou da busca dos dados armazenados

na memória do programa, sendo considerada uma limitação deste trabalho.

Na implementação de sistemas multiprocessados, cada processador armazena em sua cache uma cópia de parte do conteúdo acessado da memória para reduzir a latência no acesso aos dados e instruções. Entretanto, apesar de maximizar o desempenho dos processadores, a existência de dados replicados em cada uma das caches demanda a criação de mecanismos de coerência para garantir a execução correta da sequência de operações.

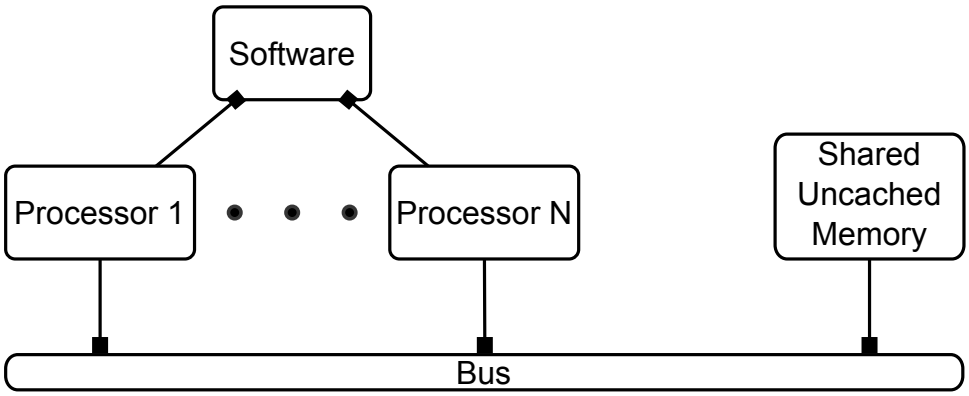


Figura 55: Organização multiprocessada de software

Apesar da limitação de modelagem de cache, o trabalho proposto suporta a especificação de comunicação e de compartilhamento de dados em plataformas multiprocessadas através da utilização de espaços de endereçamento de memória que não são mapeadas em cache, como é ilustrado na figura 55. Caso o sistema considerado precise ser alocado em endereços de memória mapeados em cache, os dados de simulação referentes as caches, como taxa de acertos ou faltas, e os tempos de simulação consumidos pelos mecanismos de coerência não serão modelados na plataforma virtual.

Na próxima subseção será descrito um exemplo de sistema com múltiplos processadores que se comunicam utilizando uma região de memória compartilhada não mapeada em cache. Para demonstrar a utilização da interface de programação concorrente, serão utilizados mecanismos de sincronismo para garantir a exclusão mútua no acesso das regiões críticas durante a execução concorrente do sistema.

### 4.5.1 Programação Concorrente

Para desenvolver uma aplicação paralela ou concorrente é preciso, durante a codificação em linguagem de programação, aplicar explicitamente construtores de execução concorrente. Este paradigma, conhecido como paralelismo em nível de thread ou Thread Level Parallelism (TLP) (41), difere do paradigma ILP que explora o paralelismo implícito entre as instruções. Quando bem projetado e adequado ao tipo de aplicação, a execução paralela das operações aumenta o desempenho do sistema, mas necessita da utilização de mecanismos de sincronismo e controle de concorrência para evitar condições de corrida que podem comprometer o comportamento do sistema.

O desenvolvimento de software para plataformas multiprocessadas é suportado neste trabalho através da interface nativa de programação POSIX (60), que é um padrão IEEE 1003 (62) amplamente adotado pelos principais sistemas operacionais. A escolha desta interface em particular não representa uma limitação, sendo possível a utilização de outras interfaces de programação concorrente que estejam disponíveis nativamente ou na arquitetura alvo.

Como foi visto nas seções anteriores deste capítulo, cada módulo de processador possui o suporte para interface POSIX e durante a inicialização do ambiente de simulação são criadas threads para cada instância da plataforma. Esta organização simula em alto nível a concorrência existente entre os núcleos de processamento e permite um melhor aproveitamento dos recursos do sistema de desenvolvimento nativo, uma vez que o ambiente de simulação é implementado de forma paralela.

#### 4.5.1.1 Definições da Plataforma

O acesso aos recursos da plataforma pelo software do sistema é feito através da criação de um cabeçalho da plataforma, contendo todas as inclusões de bibliotecas necessárias, definições e declaração de variáveis. A implementação deste código de cabeçalho deve ser independente da aplicação, podendo ser utilizado por outras aplicações que executem nesta mesma plataforma.

No código fonte 4.7 é fornecido um exemplo de como o código do cabeçalho da plataforma pode ser implementado. Este arquivo é dividido em duas

partes: arquitetura alvo (linhas 1 a 11) e modelo de sistema nativo proposto (linhas 12 a 23). Em ambos casos são utilizados os mesmos mecanismos para execução multiprocessada, sendo baseada em registradores para identificação do processador (GSMPID) e para controle de acesso de região crítica de memória através de mutex (GSMPMT).

Código Fonte 4.7: Exemplo de cabeçalho de definições da plataforma

```
1  // Real Hardware header
2  #ifdef __REAL_HW__
3  // Standard I/O include
4  #include <stdio.h>
5  // Shared Memory pointer
6  unsigned int* PTR = NULL;
7  // SMP registers
8  #define GSMPMT_BUS_ADDRESS (0xFFFFFFF4)
9  #define GSMPID_BUS_ADDRESS (0xFFFFFFF8)
10 #define GSMPMT (*((volatile unsigned int*)(GSMPMT_BUS_ADDRESS
       ))
11 #define GSMPID (*((volatile unsigned int*)(GSMPID_BUS_ADDRESS
       ))
12 // HdSC Model header
13 #else
14 // HdSC Software include
15 #include <software.hpp>
16 // HdSC wrapping software header for code instrumentation
17 #include <hsc_wrapper.hpp>
18 // Functions encapsulation
19 #define lock_item(...)    __ENCAP__(lock_item, __VA_ARGS__)
20 #define unlock_item(...)  __ENCAP__(unlock_item, __VA_ARGS__)
21 #define producer(...)     __ENCAP__(producer, __VA_ARGS__)
22 #define consumer(...)     __ENCAP__(consumer, __VA_ARGS__)
23 #endif
```

A maioria das linhas é dedicada a incluir as bibliotecas padrões e da plataforma, entretanto a declaração do ponteiro PTR (linhas 5 e 6) é feita para permitir o acesso à região de memória compartilhada da plataforma através

de palavras de 32 bits com o tipo inteiro sem sinal.

Na seção 4.3.3, pode ser visto no exemplo do código fonte 4.5 a modelagem de ponteiros, no qual foi criado um ponteiro de nome PTR. A motivação para esta modelagem de ponteiro é permitir que o acesso de posições de memória em um ambiente de simulação nativo sejam traduzidos em endereços do espaço de endereçamento da plataforma virtual, utilizando a mesma sintaxe utilizada pela linguagem de programação. Neste exemplo foi criado somente um ponteiro para acessar o dispositivo, mas a modelagem permite a criação de múltiplos ponteiros, caso seja necessário, sempre buscando atender aos requisitos da plataforma e não de uma aplicação em especial.

#### 4.5.1.2 Código Fonte da Aplicação

Para ilustrar a execução multiprocessada da aplicação, foi implementada uma aplicação clássica de produtor e consumidor (ver capítulo 2 de conceitos básicos), onde as instâncias de processadores com números de identificação pares e ímpares são produtoras e consumidoras, respectivamente. As instâncias produtoras acessam concorrentemente a uma variável de contagem de itens, que está armazenada na memória compartilhada, para incrementar o seu valor, enquanto que as instâncias consumidoras acessam concorrentemente esta mesma variável para decrementar seu valor.

No exemplo de aplicação concorrente, visto no código fonte 4.8, é feita a inclusão do cabeçalho com as definições da aplicação (linhas 1 e 2), que já foi detalhado anteriormente, a declaração de ponteiro para a região de memória compartilhada (linhas 4 e 5). Na declaração do ponteiro é definido o endereço 0x80008000 para armazenamento da quantidade de itens que será incrementado pela tarefa produtora e decrementado pela tarefa consumidora.

Código Fonte 4.8: Exemplo de código fonte de programação concorrente

```
1 // Consumer/Producer include
2 #include <consumer_producer.h>
3
4 // Item counter pointer
5 unsigned int* item = (unsigned int*)(0x80008000);
```

```
 6
 7 // Lock item function
 8 void lock_item(unsigned int id) {
 9   // Attempting to lock mutex
10   do GSMPMT = (id << 16) + 1;
11   // Repeat the lock until it is obtained
12   while((GSMPMT >> 16) != id);
13 }
14 // Unlock item function
15 void unlock_item(unsigned int id) {
16   // Releasing mutex
17   GSMPMT = (id << 16);
18 }
19 // Producer task
20 void producer(unsigned int id) {
21   // Forever loop
22   while(1) {
23     // Mutex lock
24     lock_item(id);
25     // Retrieving item quantity
26     unsigned int quantity = *PTR;
27     // Producing item
28     if(quantity < 10) quantity++;
29     // Updating item quantity
30     *PTR = quantity;
31     // Outputting information
32     printf("(Processor␣%u)(Producer)␣%u␣item(s)\n", id,
         quantity);
33     // Mutex unlock
34     unlock_item(id);
35   }
36 }
37 // Consumer task
38 void consumer(unsigned int id) {
39   // Forever loop
```

```
40    while(1) {
41      // Mutex lock
42      lock_item(id);
43      // Retrieving item quantity
44      unsigned int quantity = *PTR;
45      // Consuming item
46      if(quantity > 0) quantity--;
47      // Updating item quantity
48      *PTR = quantity;
49      // Outputting information
50      printf("(Processor␣%u)(Consumer)␣%u␣item(s)\n", id,
          quantity);
51      // Mutex unlock
52      unlock_item(id);
53    }
54  }
55  // Main function
56  int main(int argc, char* argv()) {
57    // Setting item pointer
58    PTR = item;
59    // Resetting item quantity
60    *PTR = 0;
61    // Retrieving processor ID
62    unsigned int id = GSMPID;
63    // Processor task allocation
64    if(id % 2 == 0) producer(id);
65    else consumer(id);
66    // Returning success
67    return 0;
68  }
```

Na função principal da aplicação (linhas 55 a 68) é feita a atribuição do endereço ao ponteiro PTR, a inicialização da quantidade de itens para o valor zero, a obtenção do número de identificação do processador e alocação da tarefa de produtor e de consumidor para os núcleos pares e ímpares, respectivamente.

Na tarefa de produtor (linhas 19 a 36) é executado em um laço infinito (linhas 21 e 35) que solicita o acesso exclusivo para acessar a memória compartilhada da plataforma (linhas 23 e 24) e incrementar o valor da variável de itens, garantindo que não exceda o limite de 10 unidades (linhas 25 e 30). Após incrementar este contador de itens, são exibidos no terminal de saída as informações sobre o processador, o nome da tarefa de produtor e a quantidade atual de itens disponíveis (linhas 31 e 32). Uma vez que as operações foram finalizadas, é feita a liberação da região crítica (linhas 33 e 34) para permitir que outros processadores sejam capazes de acessarem o contador de itens.

A tarefa de consumidor (linhas 37 a 54) também é executada em um laço infinito (linhas 39 e 53), repetindo o bloqueio para acessar a região crítica (linhas 41 e 42), acessando o endereço de memória que contém a variável de itens (linhas 43 e 44) e decrementando o valor do contador de itens, garantindo que não seja negativo (linhas 45 e 46). Antes de liberar a região crítica (linhas 51 e 52), o valor é atualizado na memória compartilhada (linhas 47 e 48) e a identificação do processador, o nome da tarefa de consumidor e o valor atualizado do contador de itens são exibidos (linhas 49 e 50).

Na figura 56 é exibida a saída da execução da aplicação de produtor e consumidor em uma plataforma com 4 processadores, com dois núcleos com funções de produção e dois núcleos com função de consumo. Os resultados observados são referentes as primeiras linhas geradas pela execução, uma vez que a simulação não possui critério de parada, executando indefinidamente.

```
[Processor 3][Consumer] 0 item(s)
[Processor 3][Consumer] 0 item(s)
[Processor 0][Producer] 1 item(s)
[Processor 3][Consumer] 0 item(s)
[Processor 1][Consumer] 0 item(s)
[Processor 2][Producer] 1 item(s)
[Processor 1][Consumer] 0 item(s)
[Processor 2][Producer] 1 item(s)
[Processor 0][Producer] 2 item(s)
[Processor 3][Consumer] 1 item(s)
[Processor 3][Consumer] 0 item(s)
[Processor 3][Consumer] 0 item(s)
[Processor 3][Consumer] 0 item(s)
[Processor 1][Consumer] 0 item(s)
[Processor 3][Consumer] 0 item(s)
[Processor 0][Producer] 1 item(s)
[Processor 1][Consumer] 0 item(s)
[Processor 0][Producer] 1 item(s)
[Processor 0][Producer] 2 item(s)
[Processor 0][Producer] 3 item(s)
[Processor 0][Producer] 4 item(s)
[Processor 0][Producer] 5 item(s)
[Processor 0][Producer] 6 item(s)
[Processor 0][Producer] 7 item(s)
[Processor 0][Producer] 8 item(s)
[Processor 0][Producer] 9 item(s)
[Processor 0][Producer] 10 item(s)
[Processor 0][Producer] 10 item(s)
[Processor 0][Producer] 10 item(s)
[Processor 0][Producer] 10 item(s)
[Processor 1][Consumer] 9 item(s)
[Processor 0][Producer] 10 item(s)
[Processor 1][Consumer] 9 item(s)
[Processor 1][Consumer] 8 item(s)
⋮
```

Figura 56: Execução da aplicação de produtor e consumidor

# 5 Estimativa de Tempo Simulado

Uma vez definidos os mecanismos de modelagem e de simulação de plataformas, este capítulo está dedicado a detalhar o conjunto de técnicas desenvolvidas para realização das estimativas de tempo simulado sobre a execução nativa do software. Nestas estimativas de tempo se procura medir o tempo consumido na execução nativa da aplicação e estimar qual seria o tempo simulado equivalente, caso estivesse executando em uma determinada plataforma alvo. Este processo de cálculo de tempo é suportado pela instrumentação de código fonte da aplicação, já vista anteriormente, que permite a preempção da execução da aplicação em nível de bloco básico.

## 5.1 Visão Geral

O tempo total de execução do software pode ser definido como o somatório do tempo de cada um de seus blocos básicos, e é por este motivo que, além de suportar a preempção, a instrumentação do código fonte permite que estes blocos tenham seu tempo de execução medido dinamicamente. Com o tempo medido, utilizando técnicas de mapeamento que serão descritas em detalhes nas subseções a seguir, como a média móvel de execução e o mapeamento dinâmico de tempo simulado, é possível, a partir do tempo de execução nativa, estimar o tempo simulado equivalente da aplicação sendo executada em uma determinada plataforma alvo.

A motivação para execução nativa do software reside no fato de que existe uma série de gargalos e problemas no fluxo de desenvolvimento tradicional com ISS, que, apesar de preciso, é muito lento e complexo de ser construído. Dentre as vantagens da execução nativa podem ser citados:

- Alto desempenho de simulação: como o software não precisa ser emu-

lado, ele executa diretamente sobre o hardware nativo, seu desempenho é maximizado. No extremo oposto desta abordagem está situado o ISS, que executa o software instrução por instrução, em formato binário, com precisão comparável ao hardware real. Entretanto, todo este detalhe da arquitetura do processador que precisa ser emulado, causando uma imensa quantidade de funções extras que precisam ser executadas, e com isto, o desempenho obtido é bastante reduzido;

- Ferramentas de desenvolvimento: por se tratar de software que será executado nativamente, podem ser empregados todos os recursos de desenvolvimento disponíveis, como ferramentas e ambientes de desenvolvimento nativos. Desta maneira, o processo de implementação é acelerado, uma vez que diversas ferramentas podem ser utilizadas, evitando dependências muito comuns de ferramentas exclusivas de um determinado fabricante ou fornecedor de tecnologia de sistemas;

- Abstração de plataforma: é possível desenvolver o software do sistema sem ter sido definida qual arquitetura de processador será adotada ou ainda usar uma especificação abstrata. Os resultados obtidos nas simulações podem servir como guia nestas decisões de projeto, evitando que problemas ou limitações facilmente observados em simulações não passem despercebidos. A proposta procura suportar a complexa tarefa de exploração de espaço de projeto e ajudar na escolha dos componentes mais adequados, permitindo que o projetista possa modelar o seu sistema em diferentes níveis de abstração.

Apesar da execução nativa do software da aplicação apresentar o mesmo comportamento da execução na plataforma alvo, com muitas vantagens e benefícios, existe o problema de como obter estimativas sobre o tempo de execução desta aplicação da plataforma alvo. Isto se deve a uma característica: as sequências de instruções geradas são diferentes para cada arquitetura (nativa e alvo) que possuem aspectos particulares que as diferenciam. Ou seja, para que seja possível simular o software nativamente, obtendo estas estimativas de tempo simulado da plataforma alvo, é preciso criar mecanismos que mapeiem o tempo nativo para o tempo simulado equivalente na arquitetura alvo.

## 5.2 Média de Execução

Nos estágios iniciais de desenvolvimento e experimentação, foi observado que o tempo de execução nativa de um aplicação apresenta variações em seu tempo de execução. A principal justificativa para esta variação reside no fato do sistema de desenvolvimento possuir diferentes processos não relacionados que concorrem pelo uso das unidades de processamento. O escalonamento destes processos, incluindo a execução nativa da aplicação e a simulação da plataforma, é feito de maneira não determinística, devido à impossibilidade de predição de eventos caóticos, como faltas em cache ou tempo de acesso de dispositivos. Quando esta mesma aplicação é executada em um ISS, observa-se que o seu tempo simulado é determinístico, ou seja, sempre apresenta o menos resultado após cada simulação. Desta forma, seria necessário um mecanismo que fosse capaz de amostrar o tempo de execução nativo amortizado, reduzindo os efeitos do não determinismo do sistema nativo no cálculo da média de execução.

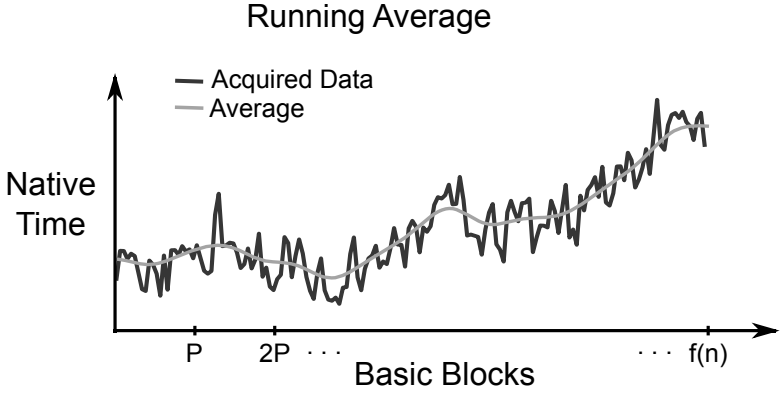


Figura 57: Amostragem de tempo de execução

Uma forma de realizar esta amostragem é aplicando a técnica de média de execução. Esta técnica consiste na interpolação dos dados obtidos e na definição de seu comportamento principal ou médio, suavizando o efeito de amostras fora de contexto, como é ilustrado na figura 57. O número total de blocos básicos da aplicação é denotado pela função $f(n)$, onde $n$ é o tamanho da entrada utilizada pela aplicação.

Neste modelo de tempo, o número de amostras coletadas entre cada estimativa de tempo é definida por um parâmetro de granularidade $P$ que define o tamanho da partição que será utilizada para o cálculo da média de

execução. Isto significa que ao invés de realizar a execução somente bloco por bloco, o núcleo de simulação é capaz de agrupar estes blocos básicos e calcular um valor médio para sua execução, melhorando a qualidade das estimativas realizadas, uma vez que mais amostras são coletadas.

$$Média(t) = \frac{1}{P} \times \sum_{i=1}^{P} Amostra(t_i) \tag{5.1}$$

Na fórmula 5.1, é observado que a $Média(t)$ é igual ao somatório das amostras de tempo $Amostra(t_i)$ dividida pelo tamanho da janela de amostragem considerada que possui o valor $P$. O motivo de se considerar este conjunto de tamanho $P$ é para ajustar o peso de cada amostra de tempo obtida, permitindo que os efeitos decorrentes do não determinismo da execução do software no ambiente nativo sejam atenuados. O valor do parâmetro $P$ deve ser definido levando em consideração a experiência do projetista do sistema para satisfazer o compromisso entre a precisão de tratamento dos eventos da plataforma (quanto menor o valor de $P$, menor é a granularidade de simulação) e a melhoria do desempenho de simulação (quanto maior o valor de $P$, menos escalonamento é necessário).

## 5.3 Mapeamento Dinâmico de Tempo Simulado

Com aplicação das técnicas de amostragem de tempo, foi possível processar as medições do tempo de execução nativo, permitindo que a partir da execução nativa do software sejam geradas estimativas de tempo simulado em um ambiente virtual de simulação. É importante destacar que o mapeamento dinâmico do tempo simulado é configurável e não possui dependências de uma determinada plataforma alvo, como será descrito nas seções a seguir.

### 5.3.1 Contexto

No estado da arte considerado, o trabalho de Wang et al. (2) trata este problema anotando estaticamente o código fonte com informações de tempo simulado obtidas através de simulação do software compilado para a plata-

forma alvo. Apesar de bem efetivo no seu objetivo, com baixas taxas de erro nas estimativas de tempo, este trabalho possui uma forte restrição: necessita que a aplicação seja compilada, e que, consequentemente, a plataforma alvo esteja definida, além de ser necessária a geração de uma base de dados com informações de tempo. Outro ponto pertinente é que sempre que modificações no código fonte da aplicação forem feitos, é necessário refazer o processo de anotação de tempo, o que para sistemas muito complexos pode se tornar impraticável, visto os longos tempos de simulação em ISS que seriam necessários.

Tendo em vista estes fatores positivos e negativos, é proposto um mecanismo de mapeamento dinâmico de tempo que anota o código fonte da aplicação em nível de bloco básico, com o uso da instrumentação de código já descrita anteriormente. Este recurso permite estimar o tempo simulado de cada bloco básico executado, sem necessidade de simulações prévias nem escolha obrigatória de arquitetura alvo de processador.

Como as estimativas de tempo simulado refletem o comportamento de uma arquitetura genérica, ou seja, sem relação com qualquer arquitetura, são necessários parâmetros para configurar as estimativas geradas e aproximar o comportamento de modelos reais. Durante o processo de concepção foi observado que dois parâmetros são suficientes para caracterizar o modelo de tempo simulado proposto, permitindo a obtenção de estimativas precisas com diferentes níveis de granularidade. Estes dois parâmetros de configuração são:

- Ajuste de tempo simulado (Tune ou T): com este parâmetro de ajuste é aplicado um fator de escala sobre a estimativa de tempo simulado gerado pela execução de um conjunto de blocos básicos. Este conjunto possui o tamanho $P$, referente a quantidade de amostras de blocos básicos utilizadas para composição da média móvel de execução. A capacidade de ajuste da escala da média obtida possibilita a amplificação ou atenuação do valor da estimativa gerada, permitindo que um atraso maior ou menor seja gerado no ambiente de simulação, respectivamente. O cálculo do valor de $T$ é gerado a partir do valor de $P$ definido pelo projetista, através do algoritmo de minimização de erro utilizado pelo resolvedor de restrições na etapa de calibração;

- Priorização de execução (Priority ou P): para permitir uma execução do

software nativo concorrente ao ambiente de simulação, é preciso definir uma priorização entre estas tarefas, como forma de organizar sua execução e permitir seu sincronismo. Este aspecto de prioridade é diretamente relacionado ao tamanho $P$ de amostras dos blocos básicos da média móvel de execução, pois quando o software está executando, a plataforma virtual está suspensa aguardando o fim da execução. Desta forma, quanto maior o valor do parâmetro $P$, maior será o tempo de execução do software sobre a plataforma e consequentemente, menor será sua granularidade.

Na subseção a seguir, o mecanismo de mapeamento será descrito em detalhes, assim como os dois parâmetros de configuração do sistema $T$ e $P$, que são os conceitos básicos para o cálculo das estimativas de tempo simulado no ambiente virtual de simulação.

### 5.3.2 Mecanismo de Mapeamento

Para o completo entendimento deste mecanismo de mapeamento, é necessário ter bem claro os conceitos de bloco básico e de como é feita a medição do tempo de execução de cada um deles. O bloco básico é uma sequência atômica de operações entre desvios, como já foi visto na instrumentação de código, podendo ser um conjunto ou partição de blocos de tamanho $P$. Uma vez definido o tamanho $P$ desta partição, é feita uma amostragem do tempo médio de execução desta partição de blocos básicos.

Esta amostragem é realizada medindo o tempo consumido pelo sistema nativo para executar as instruções contidas em cada bloco básico da aplicação, usando temporizadores padrões de alta precisão. Neste ponto, o núcleo de simulação HdSC já conhece o número de blocos básicos que simulou e qual foi o tempo nativo consumido neste processo. Utilizando as informações de tempo obtidas durante a execução nativa, é feito o mapeamento deste tempo nativo em um tempo simulado equivalente na plataforma virtual, como se o software estivesse executando em um modelo de processador com aproximação de tempo.

Antes de descrever o mecanismo de mapeamento, serão fornecidas as bases para esclarecer os conceitos em que esta técnica se baseia. Na figura

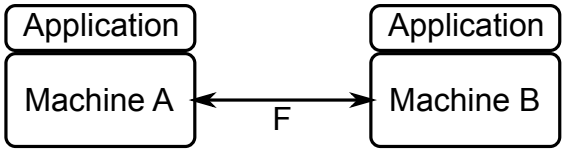


Figura 58: Conceito de mapeamento de tempo de execução

58 são ilustradas duas máquinas distintas, denotadas por $A$ e $B$ que executam uma mesma aplicação. Neste cenário, é assumido como premissa que a aplicação em questão sempre possui o mesmo tempo de execução em uma determinada arquitetura de processador, ou seja, a execução do sistema é determinística. Esta premissa é consistente com a formalização do mecanismo de mapeamento de tempo simulado que é baseada na teoria de análise de complexidade, que será descrita ainda nesta seção. Levando em consideração estas premissas e que as técnicas propostas representam uma simplificação de sistemas complexos e não determinísticos, é possível estabelecer que existe uma função linear $F$ que é capaz de estabelecer uma relação entre os tempos de execução da aplicação nas máquinas $A$ e $B$ quaisquer. Os principais benefícios destas considerações são: maximizar o desempenho de simulação, uma vez que os detalhes do sistema são abstraídos; e possibilitar a definição de uma função de estimativa de tempo simulado baseada nos parâmetros de configuração $T$ e $P$, descritos na seção anterior.

Para fazer este mapeamento é necessário que a menor unidade de atraso da plataforma virtual seja simulada, como forma de garantir que nenhum evento de simulação seja mascarado por estar em uma escala inferior ao atraso aplicado. Este menor atraso é definido pelo projetista, normalmente estando associado à operação com menor tempo consumido possível na plataforma, podendo ser o tempo da instrução mais rápida ou qualquer outra operação previamente conhecida na especificação da plataforma, de forma que idealmente este atraso gerado não cause perda de eventos na simulação da plataforma.

Na formalização destes conceitos, deve ser definido pelo projetista o parâmetro de tempo simulado em nanosegundos $TS_{Menor\ Atraso}$ que é baseado nas restrições de tempo da plataforma que está sendo simulada. Para realizar a simulação no ambiente nativo da quantidade de tempo simulado definido pelo menor atraso, para todos os componentes da plataforma, é consumida uma quantidade de tempo nativo em ciclos de processador que é denotado

por $TN_{Plataforma}$. Assumindo que existe uma relação linear $F$ entre a máquina da plataforma de simulação e a máquina de execução nativa, é feito o mapeamento do tempo de nativo para o tempo simulado na plataforma. O tempo nativo em ciclos de processamento de execução de uma sequência de blocos básicos de tamanho $P$ é denotado por $TN_{Bloco\ Básico}$ e, aplicando o mapeamento entre os tempos nativo e simulado, é derivada uma estimativa de tempo simulado em nanosegundos denotada por $TS_{Estimativa}$ que é ajustada pelo parâmetro $T$.

$$\begin{aligned} TS_{Menor\ Atraso}(ns) &\leftrightarrow TN_{Plataforma}(ciclos) \\ TS_{Estimativa}(ns) &\leftrightarrow TN_{Bloco\ Básico}(ciclos) \end{aligned} \tag{5.2}$$

Na fórmula 5.2 é feita a sistematização destes conceitos, indicando as relações entre os tempos nativo e simulado da plataforma. As variáveis de tempo nativo $TN_{Bloco\ Básico}$ e $TN_{Plataforma}$ são dinamicamente avaliadas durante a simulação para fornecer estimativas de tempo simulado com maior precisão, uma vez que o conjunto de comportamentos simulados na plataforma e a sequência de operações dos blocos básicos são definidos pelo fluxo de execução da aplicação.

$$TS_{Estimativa}(ns) = \frac{TS_{Menor\ Atraso}(ns)}{TN_{Plataforma}(ciclos)} \times TN_{Bloco\ Básico}(ciclos) \tag{5.3}$$

Com a medição do tempo de execução nativo do bloco básico da aplicação, é feita o cálculo de quanto tempo simulado deve ser estimado na plataforma virtual para representar o tempo de execução equivalente do bloco básico no ambiente de simulação, gerando a fórmula 5.3. É importante ser observado que o mecanismo de estimativa não é simplesmente a aplicação de uma fórmula direta, pois todas as amostras de tempo nativo obtidas são calculadas dinamicamente durante a execução, já discutidos neste trabalho, como a média de execução e os parâmetros de configuração.

Para melhorar o entendimento do leitor, sem acarretar em perda de generalidade, será fornecido a seguir um pequeno programa de exemplo para ilustrar os conceitos de análise de complexidade. Os conceitos de análise de complexidade são essenciais e completamente alinhados com o conceito de

bloco básico, permitindo que uma análise teórica de desempenho possa ser diretamente aplicada neste contexto.

A notação utilizada define que o tempo de execução de cada bloco básico é denotado por constantes $c_i$ e que o programa possui uma entrada de tamanho $n$. Independente do tamanho da entrada utilizada, é fato que o tempo de execução de cada bloco básico é constante, dependendo somente da arquitetura que o executa.

Código Fonte 5.1: Exemplo de aplicação

```
1 Programa(entrada n) {
2   c1;
3   for(n) {
4     c2;
5     for(n) {
6       c3;
7     }
8   }
9 }
```

No código fonte 5.1, um programa de exemplo é fornecido com três blocos básicos denotados pelas constantes $c_1$, $c_2$ e $c_3$, sendo repetidos em função do tamanho de entrada $n$, passado como argumento de entrada para a aplicação. Para calcular o tempo de execução deste programa ou sua complexidade de tempo, sem dependências de arquitetura de processador e em função do tamanho de sua entrada, é preciso que seja calculado o número total de iterações realizadas.

$$Tempo\ Programa(n) = c_1 + n\,[c_2 + (c_3 \times n)] = c \times \Theta(n^2) \equiv \sum_{i=1}^{n^2} c_i' = c \times n^2 \qquad (5.4)$$

O tempo de execução do programa em função do tamanho da entrada ou complexidade de tempo pode ser descrita na fórmula 5.4. No primeiro passo é feita uma contabilização do tempo total do programa em função de suas constantes e do tamanho de sua entrada. Para analisar a complexidade de tempo para uma aplicação considerando diferentes entradas, é preciso

calcular as situações de melhor caso (denotado por $\Omega$), quando o número de iterações é o menor possível e de pior caso (denotado por $O$), quando o máximo possível de iterações é executado. Considerando o programa fornecido como exemplo, a situação de caso médio está definida entre o melhor e o pior caso da execução que pode ser expressa como $\Omega(n) \leq Tempo\ Programa(n) \leq O(n)$. Em situações que o melhor caso e o pior são iguais, a aplicação possui uma ordem exata (denotado por $\Theta$), ou seja, independentemente dos valores utilizados na entrada, o programa irá executar o mesmo número de iterações (63).

Colocando este conceito em outras palavras, isto quer dizer que um novo valor médio de constante $c$ será iterado pelo número de $n^2$ vezes, desde que $n$ possua um valor grande o bastante para tornar desprezíveis os termos de menor grau do polinômio. Na última etapa, é definido que isto seria equivalente a dizer que é um somatório de $c_i'$ de 1 até $n^2$, que pode ser reescrito como um valor médio de constante $c$ vezes o número total de iterações $n^2$. Isto reflete exatamente como é feita a amostragem de tempo dos blocos básicos, derivando sempre uma média móvel ao final de cada execução.

$$Tempo\ Programa(n) = c \times \Theta(n^2) = c \times Complexidade \tag{5.5}$$

No exemplo de programa fornecido em 5.1, foi visto que o tempo de execução só depende do tamanho $n$ de sua entrada e que sua complexidade é sempre quadrática, considerando uma entrada suficientemente grande. Caso seja feita uma generalização, considerando um tamanho de entrada $n$ fixo, é possível afirmar que o tempo de execução de uma aplicação ou sua complexidade, qualquer que seja ela, é fixa e só dependerá das constantes associadas a plataforma na qual está executando, conforme definição da fórmula 5.5.

Uma vez definido o conceito de complexidade e de blocos básicos, é preciso definir o conceito de partição de blocos básicos e em como seu tempo é medido. Já foi visto que cada bloco básico possui um tempo de execução associado e que este tempo está associado a constantes denotadas por $c_i'$. O que a partição faz é o somatório destes tempos constantes para derivar o tempo da partição, possibilitando a execução com maior granularidade.

$$Tempo\ Partição(P) = \sum_{i=1}^{P} c'_i = c_m \times P \tag{5.6}$$

Na fórmula 5.6, o parâmetro de tamanho da partição ou de configuração de prioridade $P$ é relacionado com o tempo de execução dos blocos básicos $c'_i$. Este valor de $P$ define o tamanho do conjunto de blocos básicos que serão executados, como se fossem apenas um único bloco, criando uma partição que agrupa seus tempos de execução amostrados. Desta forma, é possível concluir que a média de sua execução pode ser definida por um valor médio $c_m$ multiplicado pelo tamanho da partição.

$$Tempo\ Programa(P) = \sum_{i=1}^{\frac{Complexidade}{P}} c'_i = c \times \frac{Complexidade}{P} \tag{5.7}$$

Considerando a aplicação inteira com partições de tamanho $P$ e um número total de iterações definidas por $Complexidade$, deriva-se a fórmula 5.7, que representa o tempo de execução da aplicação com o particionamento dos blocos básicos. Verificando-se o caso base, o valor de $P$ é igual a $1$, ou seja, o conjunto de blocos básicos só possui um único bloco básico e se observa exatamente o mesmo resultado obtido na fórmula 5.4. Para os demais valores superiores a $1$ até $Complexidade$, o número total de iterações vai sendo divido por $P$, sem alterar a complexidade de tempo, apenas colocando a análise de tempo sob outra parametrização.

$$Tempo\ Partição(P) = \sum_{i=1}^{P} c'_i + TS_{Menor\ Atraso} \tag{5.8}$$

Na descrição do mecanismo de mapeamento do tempo nativo para o tempo simulado, é definida a necessidade de um valor referente ao menor atraso da plataforma em questão. Cada vez que uma partição é executada, é preciso mapear o tempo nativo em um tempo simulado equivalente e para isto é preciso gerar um atraso mínimo de tempo simulado na plataforma para fazer a estimativa. Este tempo consumido precisa ser contabilizado no cálculo do tempo da partição, conforme pode ser visto na fórmula 5.8.

$$Tempo\ Programa(P) = \frac{Complexidade}{P} \times \left( \sum_{i=1}^{P} c'_i + TS_{Menor\ Atraso} \right) \tag{5.9}$$

Na fórmula 5.9, é feita a incorporação do tempo da partição no tempo do programa, considerando o menor atraso necessário para a estimativa de tempo. Isto é feito substituindo o valor médio constante $c$ pela expressão de tempo da partição que representa o conjunto de blocos básicos considerados. Esta estimativa de tempo é referente ao tempo total simulado consumido pelo programa, durante sua execução na plataforma virtual. Observa-se que independentemente do valor de $P$, o mesmo número de blocos básicos serão executados e a complexidade da aplicação permanece constante, em função do tamanho $n$ de sua entrada, assumindo que $n$ possui um valor suficientemente grande.

É importante ressaltar mais uma vez que uma aplicação possui uma complexidade que independe da arquitetura que a executa e este custo computacional é totalmente definido em função do tamanho $n$ da entrada utilizada. Assim, é correto afirmar que uma aplicação que se utiliza de uma entrada de tamanho fixo, terá sempre o mesmo número de iterações e portanto com uma complexidade invariável.

$$Tempo(P,T) = \left[ \frac{Complexidade}{P} \times \left( \sum_{i=1}^{P} c'_i + TS_{Menor\ Atraso} \right) \right] \times T \tag{5.10}$$

Sabendo que a complexidade de uma aplicação é constante, quando uma entrada de tamanho constante é utilizada, é possível considerar este fator uma constante e tornar a função de tempo independente do parâmetro $n$, que é considerado constante. Na fórmula 5.10, a complexidade é tratada como uma constante e um novo parâmetro $T$, que já foi discutido anteriormente, é anexada à função de tempo simulado do programa. Este fator de ajuste $T$ possui um efeito de escala nas estimativas realizadas, podendo amplificar ou atenuar seus efeitos, funcionando assim para definir a granularidade temporal de cada partição executada, aumentando ou reduzindo o atraso estimado na plataforma.

$$Tempo(P,T) = \left[ \frac{Complexidade}{P} \times (c \times P + TS_{Menor\ Atraso}) \right] \times T \tag{5.11}$$

Aplicando a análise de tempo de execução da partição (fórmula 5.6) no tempo total estimado (fórmula 5.10), é obtida a fórmula 5.11. Nesta equação, o tempo médio das partições de blocos básico é colocado em termos de sua média móvel de execução $c$, sendo multiplicada pelo tamanho $P$ da partição. Este passo é considerado uma reorganização e incorporação do aspecto de média de execução na função de tempo simulado do programa.

$$Tempo(P,T) = \left[Complexidade \times \left(c + \frac{TS_{Menor\ Atraso}}{P}\right)\right] \times T \tag{5.12}$$

Reorganizando os fatores da fórmula 5.11, é obtida a fórmula 5.12. É importante que seja enfatizado que a complexidade, o valor médio do bloco básico e o menor atraso são constantes do programa e da plataforma, respectivamente. Isto quer dizer que uma aplicação não varia sua complexidade, o tempo médio de execução dos blocos básicos é constante e o menor atraso da plataforma é fixo e conhecido em sua especificação.

$$Tempo(P,T) = \left(\frac{K_1}{P} + K_2\right) \times T \tag{5.13}$$

Com todas estas informações e definições, mais uma vez os fatores da fórmula 5.12 são reorganizados para derivar a fórmula 5.13. Basicamente, todos estes fatores constantes são agrupados nas constantes $K_1$ e $K_2$ que representam as características da plataforma virtual e da aplicação combinadas, respectivamente, servindo como modelo de tempo simulado para toda a plataforma. São necessárias pelo menos duas simulações para a resolução do sistema de variáveis, que é a etapa de calibração do sistema, e assim estimar, sem necessidade de outras simulações, o tempo simulado obtido com determinados valores de $P$ e $T$.

A principal consequência deste modelo teórico de estimativa de tempo simulado é a possibilidade de uma aplicação, com entrada e plataforma de execução conhecidas, possa ter seu comportamento temporal previsto estaticamente, necessitando somente de poucas simulações para identificação das constantes associadas a plataforma e a aplicação em questão. Os próximos passos consistem em colocar este modelo a prova e realizar a análises experimentais, como forma de demonstrar sua eficiência e precisão em estimativas de tempo simulado.

### 5.3.3 Análise Experimental

Com a formalização do mecanismo de mapeamento de tempo nativo para tempo simulado, detalhado na seção anterior, são realizadas análises experimentais para colocar a prova o comportamento teórico previsto. Para tanto, é escolhido um aplicativo de referência chamado Dhrystone 2.1, que é muito utilizado em avaliação de desempenho de processadores, com tamanho de entrada fixo que representa o número de iterações a serem realizados e dispositivos na plataforma para medição de tempo.

Para obtenção das curvas de estimativa de tempo simulado são feitas simulações com diferentes valores para os parâmetros de priorização $P$ e de ajuste $T$. O objetivo é realizar a coleta de dados estimados e se analisar o comportamento temporal da aplicação. Como já foi dito, o conceito chave da fórmula 5.13 é prever estaticamente o comportamento temporal estimado, realizando poucas simulações para determinar o valor das constantes $K_1$ e $K_2$.

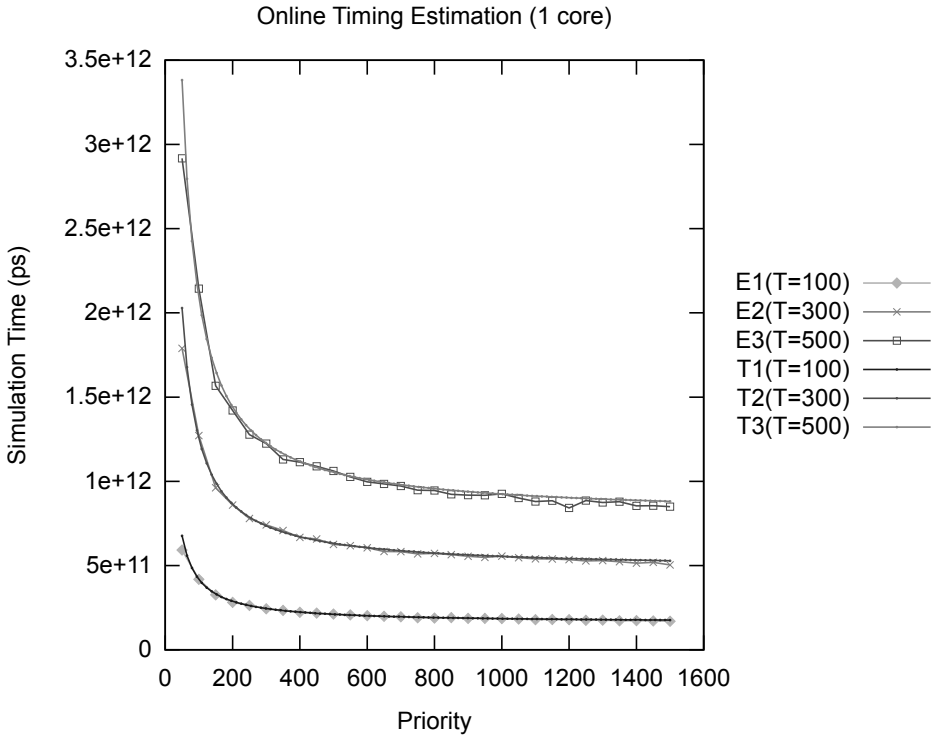


Figura 59: Tempo de simulado com parâmetros de priorização e ajuste para 1 núcleo

Na figura 59, o gráfico exibe três curvas com diferentes fatores de ajuste $T$ e com diferentes priorizações $P$ para cada curva no eixo horizontal. Observando o eixo vertical, as estimativas dos tempos de simulados (Simulation Time)

são exibidas em uma escala de picosegundos (ps). Cada uma destas curvas é o resultado de diferentes valores do parâmetro de ajuste $T$, causando uma translação de nível da curva no gráfico, confirmando o efeito de escala previsto no modelo teórico. Este ajuste $T$ permite que, para um mesmo tamanho de partição, diferentes atrasos sejam estimados, ajustando assim o nível de granularidade temporal obtido.

Os efeitos do não determinismo da simulação nativa podem ser percebidos nas variações de amostragem das curvas experimentais $E1(T = 100)$, $E2(T = 300)$ e $E3(T = 500)$. Estas variações já haviam sido identificadas em estágios iniciais e todos os mecanismos de filtragem para estes efeitos apresentam um comportamento satisfatório, principalmente por estar trabalhando em uma escala de alta resolução em picosegundos. Nas curvas teóricas $T1(T = 100)$, $T2(T = 300)$ e $T3(T = 500)$, calculadas estaticamente com os valores das constantes $K_1 = 258683915583,33$ e $K_2 = 1590087775,41$, são exibidas previsões de estimativas de tempo simulado com os mesmos parâmetros das curvas experimentais.

É possível ver no gráfico 59 que os valores de tempo simulados estimados atendem ao comportamento previsto no modelo teórico e são coerentes com os resultados obtidos nas simulações realizadas, tanto no desenho dos pontos das funções teóricas, como nas curvas experimentais obtidas por simulações. Logo, o resultado experimental observado confirma o comportamento do modelo de estimativa de tempo, tanto em simulações dinâmicas, como em previsões estáticas, com taxas de erro inferiores a 1%.

Apesar de um tratamento formal e de um estudo de caso para confirmar seu funcionamento, está dedicado o capítulo 6 de resultados para explorar este modelo de estimativas de tempo simulado. Serão aplicadas diferentes classes de aplicações, com diferentes parâmetros de priorização $P$ e de ajuste $T$, como forma de se verificar o erro das estimativas e qual a melhoria de desempenho da abordagem proposta.

## 5.4 Estimativas em Plataformas Multiprocessadas

Independentemente do número $N$ de núcleos de processamento ou de seu arranjo, as estimativas de tempo simulado realizadas seguem os mesmos

critérios já descritos anteriormente. Isto se deve ao fato das especificações do sistema serem modeladas sem ligação com o número de núcleos de processamento que serão utilizados. Portanto, quando as restrições de tempo são aplicadas, elas dizem respeito ao sistema completo, abstraindo o número de processadores utilizados.

$$Tempo(P,T) = \left(\frac{K'_1}{P} + K'_2\right) \times T \tag{5.14}$$

$$K'_1 = \sum_{i=1}^{N} K_{1_i} \quad , \quad K'_2 = \sum_{i=1}^{N} K_{2_i} \tag{5.15}$$

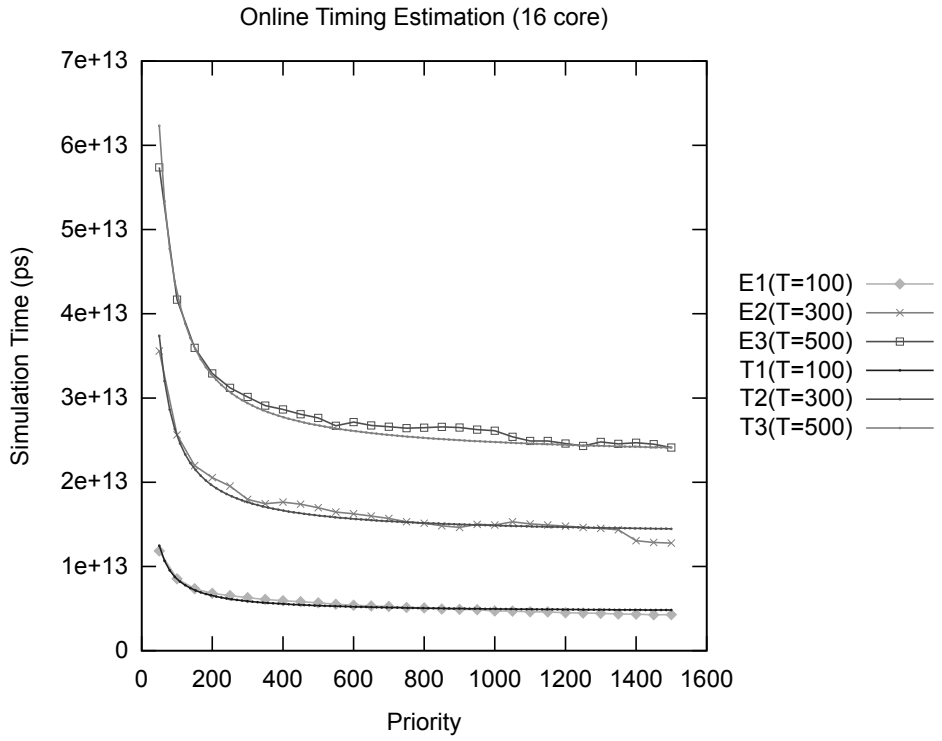


Figura 60: Tempo de simulado com parâmetros de priorização e ajuste para 16 núcleos

As fórmulas 5.14 e 5.15 formalizam este comportamento e esta característica de abstração confere ao projetista uma capacidade de avaliar o sistema e seu comportamento como um todo, criando um foco na concepção de uma arquitetura de plataforma que atenda aos requisitos do projeto.

Para avaliar este comportamento teórico previsto, foram realizados experimentos com a aplicação de referência Dhrystone 2.1 em uma plataforma com 16 núcleos de processamento. Na figura 60 podem ser vistas três curvas

experimentais $E1(T = 100)$, $E2(T = 300)$ e $E3(T = 500)$ obtidas a partir de resultados de simulações utilizando diferentes configurações dos parâmetros de sistema $P$ e $T$.

De forma sobreposta a esta figura 60 são desenhadas mais três curvas teóricas: $T1(T = 100)$, $T2(T = 300)$ e $T3(T = 500)$ que foram obtidas através da determinação dos valores das constantes $K'_1 = 3951933415160,67$ e $K'_2 = 45587475684,53$ do modelo de tempo do sistema, em conformidade com o comportamento teórico previsto e gerando estimativas de tempo simulado com erro médio inferior a 2%, com relação a referência baseada em ISS.

# 6 Resultados

Este capítulo é focado na exposição dos resultados obtidos por este trabalho proposto, além de realizar sua contextualização no estado da arte considerado. Na primeira seção, a plataforma de referência utilizada nos estudos de caso é definida e ilustrada, permitindo ao leitor uma visão clara dos recursos de hardware e software disponíveis. Após definir a plataforma e seus dispositivos, as métricas de desempenho e o conjunto de aplicações são listados e categorizados. Para cada aplicação é feita a descrição de suas funcionalidades e exibição do seu comportamento durante sua execução no modelo proposto, exibindo seu grau de precisão e desempenho obtidos.

Uma vez que todos os experimentos foram realizados e os dados foram obtidos, é realizada uma análise comparativa entre os resultados obtidos e o estado da arte. A ideia desta seção é trazer ao leitor o impacto deste trabalho proposto, no contexto em que ele se propõe a gerar contribuições. Desta maneira, é desejado que no final desta análise ficassem evidentes as melhorias atingidas, assim como as limitações inerentes desta abordagem proposta.

## 6.1 Plataforma de Referência

Como ponto de partida para realização dos estudos de caso, foi necessário conceber e construir uma plataforma virtual de referência para realização dos experimentos. Durante a análise do que seria necessário para definir a plataforma, foram enumerados os seguintes requisitos:

- Sempre utilizar especificações de componentes reais de hardware, como forma de garantir a eficácia dos estudos de caso. Ou seja, todos os componentes utilizados nesta plataforma são uma implementação aderente a uma especificação de um componente comercial, podendo inclusive

suportar software binário desenvolvido para sua utilização;

- Permitir a exploração das capacidades propostas de acesso e controle dos dispositivos da plataforma. Foram previstos um barramento de interconexão baseado em transações e quatro dispositivos de captura e exibição de imagem, além de controle para contagem e medição do tempo. Além do propósito validação de conceito, os componentes utilizados são essenciais para o correto funcionamento das aplicações;

- A escolha do processador deve ser determinada pela disponibilidade de ferramentas de desenvolvimento e depuração, além do próprio ISS. Este ISS deve ser capaz de ser integrado em uma plataforma virtual e ser capaz de acessar e ler seu programa de dispositivos disponíveis na plataforma. Além de todos estes aspectos técnicos, sua arquitetura deve ter relevância em aplicações comerciais, para acreditar os resultados obtidos com a arquitetura considerada;

- A plataforma deve ser flexível e escalável, permitindo que novos dispositivos sejam adicionados e utilizados sem impactar na configuração préexistente. Um ponto importante na escalabilidade é o suporte para múltiplas instâncias de processadores, para execução de software com multiprocessamento, sem causar reprojeto ou modificações drásticas no projeto inicial. A ideia principal deste requisito é realizar escolhas que não limitem o potencial de uso futuro desta plataforma.

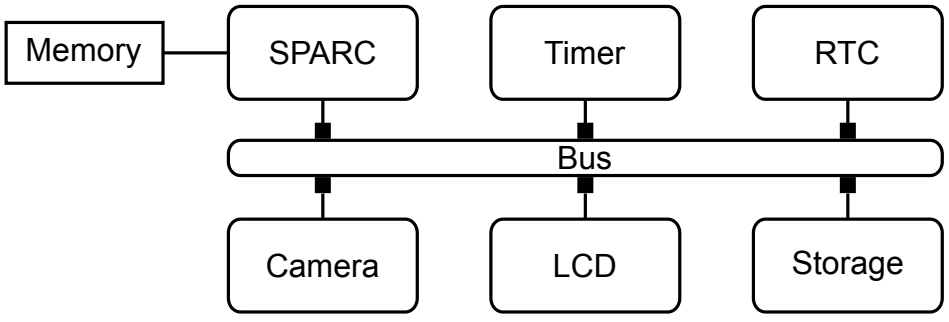


Figura 61: Plataforma de referência

Na figura 61, a plataforma considerada é composta por um processador SPARC-V8 (SPARC) (64), acoplado a uma memória de programa (Memory), além dos seguintes periféricos:

- Barramento de interconexão TLM (Bus) operando com 32 bits de largura para dados e endereçamento, com frequência de 1 GHz;
- Timer configurável com até 8 unidades independentes (Timer) com relógio base padrão de 100 MHz;
- Contador de tempo real (RTC) com registradores de 32 bits de contagem e escala em segundos, com resolução de 100 nanosegundos;
- Sensor de imagem Aptina MT9V011 (Camera) com resolução de 640 linha por 480 colunas e capacidade de captura de até 30 quadros por segundo, operando com frequência de 27 MHz;
- Módulo de LCD Sharp LQ043T3DX02 (LCD) com resolução de 480 colunas por 272 linhas com interface RGB serial e 24 bits de definição de cores, funcionando com taxa de 9 MHz;
- Unidade de armazenamento de dados Micron MT28F320J3 (Storage) de alto desempenho e capacidade de 32 MB, com interface de largura de 32 bits para dados e endereço, operando com um tempo de acesso aleatório entre 25 e 110 nanosegundos.

O processador baseado em ISS executa em nível funcional, executando 2 instruções por ciclo de relógio, modelo de cache de dados e de instruções com taxa de acerto entre 90% e 95% de latência de 1 ciclo de relógio, com penalidade de 10 ciclos para falta de dados e frequência de operação de 1 GHz, realizando suas operações de acesso de dados e interrupção através de interface TLM. Os periféricos que demandam fluxo de dados realistas, como o sensor de imagem e o módulo de LCD, foram modelados utilizando técnicas de Hardware-in-the-Loop (HIL) (65, 66, 67) que consiste em utilizar dispositivos de hardware para realização da captura ou exibição da imagem, respectivamente. Esta plataforma foi baseada no projeto do Centro Espacial Europeu de Pesquisa e Tecnologia (ESTEC) (68), Agência Espacial Europeia (ESA) (69) e Centro de Pesquisa Gaisler (70), permitindo licenciamento aberto para utilização para fins não comerciais. Esta escolha contempla os requisitos de aplicação comercial, disponibilidade de ferramentas e de relevância em aplicações reais, que neste caso são as aeroespaciais. Os dispositivos implementados são aderentes às especificações fornecidas, como forma de suportar o software

disponível, mas por questões de escopo, apenas alguns dos periféricos serão implementados.

Outro ponto de apoio nesta escolha foi a disponibilidade de ISS do projeto ArchC (71), compatível com a arquitetura SPARC-V8, sendo totalmente integrável com o ambiente de plataforma virtual e com código aberto. Esta flexibilidade é fundamental para personalização e ajustes necessários na plataforma, seja para melhorar o desempenho ou realização de ajustes. Além desta disponibilidade, diversos dispositivos de hardware também estão disponíveis, seja através de sua especificação como por sua implementação em linguagem de descrição de hardware. Todos os componentes, principalmente o ISS, suportam adequadamente o uso em um contexto de multiprocessamento, sem limitações arquiteturais para sua implementação. Desta forma, a flexibilidade e escalabilidade estão asseguradas para expansões ou melhorias futuras.

## 6.2 Metodologia dos Experimentos

Para realizar compilação de todos os componentes da plataforma, incluindo o software, foi utilizado o GNU Compiler Collection (GCC) (72) de versão 4.2.1 para C e C++, sem utilização de diretivas para otimização de código. A execução dos experimentos dos estudos de caso foram realizados de forma totalmente automatizada, através do uso de scripts de automatização e de ferramentas de processamento de texto. Esta funcionalidade foi construída para oferecer maior confiabilidade e agilidade na execução dos experimentos, excluindo o fator humano do processo e evitando erros durante a coleta das informações, seja para geração de configurações de parâmetros ou dos gráficos das curvas, por exemplo.

### 6.2.1 Simulação e Coleta de Dados

Os experimentos são realizados com a execução de todos os estudos de caso, utilizando a plataforma de referência baseada em ISS, e a coleta das informações de simulação geradas, como tempo simulado e desempenho obtido em métricas MCPS e MIPS, além de quaisquer outras informações que

sejam relevantes para a aplicação em questão.

Com todas estas informações obtidas do modelo ISS, é iniciada a fase de calibração que utiliza as informações geradas pela plataforma baseada em ISS para geração dos parâmetros de ajuste e de tempo. Nesta etapa de calibração são necessárias pelo menos 2 simulações com o modelo proposto para definição do nível de ajuste e pelo menos 2 simulações para caracterização das constantes da curva. Este processo está sendo realizado através de interpolação, aproveitando a função de tempo simulado formalizada pelo modelo de tempo, entretanto é possível a utilização de técnicas mais sofisticadas para que se obtenha uma melhor amostragem dos pontos gerados na curva.

$$Média(\mu) = \frac{1}{n} \times \sum_{i=1}^{n} Amostra(t_i) \tag{6.1}$$

$$Desvio\ padrão(\sigma) = \sqrt{\frac{1}{n} \times \sum_{i=1}^{n} [Amostra(t_i) - Média]^2} \tag{6.2}$$

Uma vez determinados os parâmetros de configuração para cada aplicação, são realizadas 30 simulações para obtenção de resultados muito próximos a uma distribuição normal (73). Com estas amostras preliminares, é calculada uma estimativa confiável da média das amostras e do desvio padrão, descritas pelas fórmulas 6.1 e 6.2, respectivamente. Para determinar a quantidade de amostras $n$, que precisam ser coletadas, é preciso definir o nível de confiança $z$ para que um intervalo contenha as amostras representativas para o cálculo da média, com valores que usualmente possuem níveis de confiança de 90%, 95% ou 99%. Outro parâmetro necessário para o cálculo de $n$ é a margem de erro $E$, que define o erro da média obtida pela amostragem.

$$Quantidade\ de\ amostras(n) = \left(\frac{z \times \sigma}{E}\right)^2 \tag{6.3}$$

Na fórmula 6.3 é definido quão grande deve ser a quantidade de amostras $n$ para que seja obtido um determinado nível de confiança $z$ nos resultados e uma margem de erro $E$ das amostras realizadas, utilizando uma estimativa de desvio padrão $\sigma$ obtida de simulações realizadas. Levando em consideração o compromisso entre a qualidade dos resultados e a quantidade de simula-

ções necessárias, foi definida nos experimentos realizados a utilização de um nível de confiança de 95%, que possui um valor de $z$ igual à 1,96 em uma distribuição normal (73), e de uma margem de erro para amostragem de 1%.

### 6.2.2 Comparação dos Resultados

No capítulo 3 de estado da arte foram descritos em detalhe uma série de trabalhos relacionados a esta abordagem proposta, permitindo ao leitor situar os conceitos e, principalmente, quais as contribuições geradas. Ao final deste capítulo, é feito um resumo dos resultados experimentais através de uma análise comparativa onde métricas de melhoria de desempenho e precisão são listadas sistematicamente em uma tabela. Apesar dos diferentes paradigmas empregados, todos os trabalhos que realizam comparação de precisão dos resultados se utilizam de uma plataforma de referência com alto nível de precisão, como um modelo RTL, um simulador baseado em ISS ou o próprio hardware real. Outro aspecto visto são os diferentes níveis de detalhe dos modelos de sistema, permitindo diferentes graus de precisão e de melhoria de desempenho.

Para verificar e validar a abordagem proposta, a plataforma de referência e o nível de abstração foram escolhidos levando-se em conta o caso médio em termos de modelagem de sistemas, embutindo o máximo possível de detalhe, mas mantendo os modelos independentes de arquitetura. Por isto, na realização de comparações com o estado da arte, o leitor deve sempre considerar os resultados de modelos com menor nível de abstração e taxa de erro. Por possuírem um maior detalhamento do funcionamento, necessário para execução de software dependente do hardware, estes modelos possuem um menor índice de melhoria de desempenho com relação à plataforma de referência.

## 6.3 Métricas de Desempenho e Precisão

As métricas de desempenho e precisão são adquiridas e avaliadas através do relatório de simulação, contendo, entre outras informações, o tempo simulado e o tempo de execução da simulação. Para este trabalho serão consideradas duas métricas de desempenho: o número de ciclos simulados por se-

gundo (MCPS) e o número de instruções executadas por segundo (MIPS), para cada processador da plataforma. Em cada um dos casos, que serão detalhados a seguir, uma determinada quantidade de operações foi realizada, seja em termos de ciclos simulados ou de instruções executadas.

Quando se fala em tempo simulado está se fazendo referência a quantidade total de tempo que foi contabilizado na plataforma virtual, considerando o somatório de todos os eventos que foram simulados. A outra medida de tempo, colocada em unidade de segundos, é referente ao tempo de execução da simulação consumido pelo computador para simular o comportamento da plataforma virtual, ou seja, o tempo de execução da simulação é o tempo real gasto pela máquina nativa.

$$MCPS = \frac{Número\ de\ Ciclos \times 10^{-6}}{Tempo\ de\ Execução}(ciclos/s) \tag{6.4}$$

Uma vez feitas estas definições, é hora de observar como os trabalhos relacionados avaliam os parâmetros de desempenho e precisão, além de explicitar como é feita a análise neste trabalho proposto. No trabalho relacionado de Schirner et al. (11), o tempo simulado e o tempo de execução são utilizados para derivar a métrica de desempenho de ciclos de relógio por segundo (ciclos/s). Nesta medição é verificada quantos ciclos são simulados (tempo simulado) em um tempo real de execução. Como normalmente esta quantidade de ciclos é muito alta, é feita uma medição com milhões de ciclos por segundo ou MCPS (Million of Cycles Per Second), conforme definição na fórmula 6.4.

$$MIPS = \frac{Número\ de\ Instruções \times 10^{-6}}{Tempo\ de\ Execução}(instruções/s) \tag{6.5}$$

Já no trabalho de Wang et al. (2), é utilizada a métrica de contabilizar o número de instruções por tempo de execução (instruções/s). Dado o grande volume de instruções executadas, esta métrica é condensada em milhões de instruções por segundo ou MIPS (Million of Instructions Per Second), conforme definição da fórmula 6.5. Em modelos ISS é um resultado natural da simulação, uma vez que o sistema executa uma sequência de instruções armazenadas em uma memória binária e no final da simulação é feita a divisão deste contador de instruções pelo tempo de execução total da plataforma.

$$MIPS = \frac{\left(\frac{Tempo\ Simulado(ns)\times 10^{-6}}{CLK(ns)\times CPI}\right)}{Tempo\ de\ Execução}(instruções/s) \tag{6.6}$$

Neste trabalho, ambas as métricas de desempenho são adotadas, sendo derivadas das definições de ciclo de relógio do processador (CLK) que é de 1 nanosegundo e do número de ciclos de instrução executados por ciclo de relógio ou CPI (Cycles Per Instruction) possui valor de $1/2$. O leitor crítico pode estar se questionando como um modelo de alto nível, como este que está sendo proposto, é capaz de gerar informações sobre o número de instruções executadas. Na fórmula 6.6 surge a resposta para esta questão, visto que com o tempo simulado estimado, juntamente com o valor do período de relógio (CLK) e o número de ciclos por instruções (CPI), é possível saber exatamente quantas instruções foram executadas.

$$Erro = \frac{|HdSC - ISS|}{ISS} \times 100 \tag{6.7}$$

A realização de estimativas de desempenho, seja com a métrica MIPS ou MCPS, é fundamental o nível de precisão obtida. Só é possível verificar o erro de uma estimativa quando é gerado um valor estimado e feita sua comparação com um valor de referência, que neste caso é o resultado do modelo ISS. Como o número de instruções e o tempo simulado são essencialmente a mesma coisa, a análise de precisão calcula qual o erro, considerando o módulo da diferença entre a estimativa proposta (HdSC) e o valor de referência (ISS), como pode ser visto na fórmula 6.7. Este erro calculado é dividido pelo resultado do ISS e multiplicado pelo valor 100, gerando um percentual de erro absoluto entre a estimativa obtida pelo trabalho proposto HdSC e o valor de referência obtido pelo ISS. Durante a exibição dos resultados obtidos serão fornecidos valores de erro médios e seus respectivos desvios padrões, como forma de demonstrar ao leitor a qualidade das estimativas geradas e qual o seu comportamento em uma análise estatística dos dados coletados.

$$Desempenho_{MCPS} = \frac{HdSC_{MCPS}}{ISS_{MCPS}} \tag{6.8}$$

$$Desempenho_{MIPS} = \frac{HdSC_{MIPS}}{ISS_{MIPS}} \tag{6.9}$$

Foi descrito como o desempenho é medido, mas ainda não foi explicitado como será feita a análise comparativa de desempenho entre o trabalho proposto (HdSC) e a abordagem tradicional baseada em ISS. Ambas as métricas de MCPS e MIPS serão utilizadas, ficando a cargo do leitor avaliar qual medida é mais significativa para comparação de desempenho. Nas fórmulas 6.8 e 6.9, é feita a divisão do desempenho obtido pelo trabalho proposto (HdSC) pelo desempenho do modelo ISS, gerando um fator de diferença multiplicativo de desempenho entre as abordagens consideradas.

É muito importante ressaltar que este trabalho objetiva a abstração do modelo de processamento, sem alterar seu comportamento original obtido no ISS e minimizando os erros gerados pelas estimativas. Por isto, em todas as aplicações utilizadas serão descritos os resultados obtidos no ISS, como tempo simulado total e desempenho alcançado. Com estas informações serão utilizados valores de parâmetros de configuração do modelo de estimativa para maximizar o desempenho da simulação e, ao mesmo tempo, minimizar o erro de estimativa de tempo obtido. Assim, é esperado que uma implementação de software se comportasse de forma equivalente em ambos os modelos, apresentando melhorias significativas de desempenho e baixo erro de precisão, quando comparado ao ISS.

## 6.4 Aplicações

Para realização dos experimentos e análises deste trabalho proposto, foram desenvolvidas e utilizadas aplicações, para avaliação dos diversos aspectos da plataforma. Para melhorar o entendimento do leitor, as aplicações foram categorizadas em três tipos:

- Entrada e saída intensiva: refere-se ao conjunto de aplicações que manipulam grande volume de dados, através de operações de leitura e escrita em dispositivos externos, como periféricos e unidades de armazenamento. Seu desempenho está diretamente ligado a capacidade de comunicação entre os componentes de hardware e software da plataforma, seja pela eficiência do barramento de interconexão ou pela limitação temporal do dispositivo em questão;

- Computação intensiva: esta classe de aplicações dedica a maior parte do seu tempo de execução para realização de operações de processamento, com pouca ou nenhuma demanda por operações de entrada e saída. Este tipo de aplicação é diretamente beneficiada pelo aumento da capacidade computacional, seja por melhoria de desempenho do processador ou pelo uso de unidades dedicadas de processamento;

- Multiprocessada: são aplicações explicitamente desenvolvidas para aproveitar mais de uma unidade de processamento disponível na plataforma. Estas aplicações devem ter a habilidade de alocar suas tarefas nos diferentes núcleos de processamento e serem capazes de lidar com os diferentes problemas decorrentes do paralelismo, como sincronização e compartilhamento de recursos.

Nas subseções a seguir, cada um destes tipos de aplicação será detalhado, com informações sobre seu comportamento e implementação. Além destas informações, serão fornecidas informações referentes aos experimentos realizados com as técnicas propostas e comparação com resultados obtidos pelo modelo ISS. O leitor deve estar atento a possíveis diferenças entre os gráficos das curvas e as parametrizações obtidas que são decorrentes dos processos adicionais utilizados para captura de dados de simulação em tempo de execução.

### 6.4.1 Entrada e Saída Intensiva

Esta subseção é dedicada às aplicações de entrada e saída intensivas, fornecendo detalhes sobre seu funcionamento e sobre os dispositivos que realizam as suas operações de entrada e saída. Os experimentos foram concebidos para explorar as diferentes características desta classe de aplicação e utilizaram o seguinte conjunto de aplicações listadas abaixo:

- Camera Capture: captura quadros de imagem no tamanho de 640 linhas por 480 colunas e formato RGB do sensor de imagem da plataforma. Após o quadro estar armazenado pela aplicação, é feito um redimensionamento da imagem utilizando procedimentos de software para exibição da imagem no módulo LCD que possui resolução de 480 linhas e

272 colunas. O conceito chave desta aplicação é demonstrar uma aplicação que realiza uma sequência de operações de entrada e saída intensiva, mas que também realiza operações de redimensionamento das imagens;

- Device Control: demonstra o controle e uso dos módulos de Timer disponíveis na plataforma, sendo geradas interrupções de hardware distintas que serão tratadas por rotinas de tratamento de interrupção específicas do software. O objetivo principal desta aplicação é ilustrar o comportamento do mecanismo de tempo simulado proposto em um cenário onde a aplicação possui um tempo fixo de execução, invariante com relação a velocidade em que o processador executa, além de demonstrar os mecanismos de preempção do software e de execução das rotinas de tratamento;

- Hello World: é uma aplicação clássica muito utilizada em cursos de linguagens de programação para ilustrar de forma bem simples o funcionamento de uma aplicação. Sua única função é exibir uma mensagem de "Hello World!" para ser exibida no terminal de execução, por isto sendo classificada como entrada e saída intensiva, por somente realizar uma operação de saída. Este programa representa um caso base ou pior cenário de utilização para a abordagem proposta, uma vez que somente um bloco básico é executado, o que reduz a precisão das estimativas de tempo geradas;

- LCD Pattern: nesta aplicação uma série de padrões de gradientes RGB e de fractais são renderizados totalmente em software para serem exibidos no módulo de LCD da plataforma, que possui imagem de tamanho de 480 linhas e 272 colunas. O conceito explorado por esta aplicação é utilizar processamento mais intensivo associado com geração massiva de dados para o dispositivo de saída de LCD, permitindo avaliar um cenário de uso híbrido de processamento e acesso externo de dispositivos;

- Mergesort: este algoritmo de ordenação em implementação é utilizado para ordenar uma grande sequência de números contidos na unidade de armazenamento da plataforma. O objetivo da escolha desta aplicação é demonstrar o funcionamento do software acessando intensivamente um dispositivo com tempo de acesso aleatório, onde a interface

de programação utilizada é baseada em ponteiros de programação que é suportada pela modelagem proposta.

As subseções a seguir irão aprofundar as informações cada uma destas aplicações, através de uma descrição detalhada, os resultados obtidos na análise experimental e as respectivas conclusões geradas por cada estudo de caso, sempre buscando contextualização com o estado da arte e com cenários distintos de utilização.

#### 6.4.1.1 Camera Capture

**Descrição** Como já foi descrito na plataforma de referência, são utilizados os periféricos: um sensor de imagem Aptina MT9V011 (Camera) com resolução de 640 linha por 480 colunas, operando com até 30 quadros por segundo e frequência de operação de 27 MHz; e um módulo de LCD Sharp LQ043T3DX02 (LCD) com resolução de 480 colunas por 272 linhas com interface RGB serial e 24 bits de definição de cores, funcionando com taxa de operação de 9 MHz.

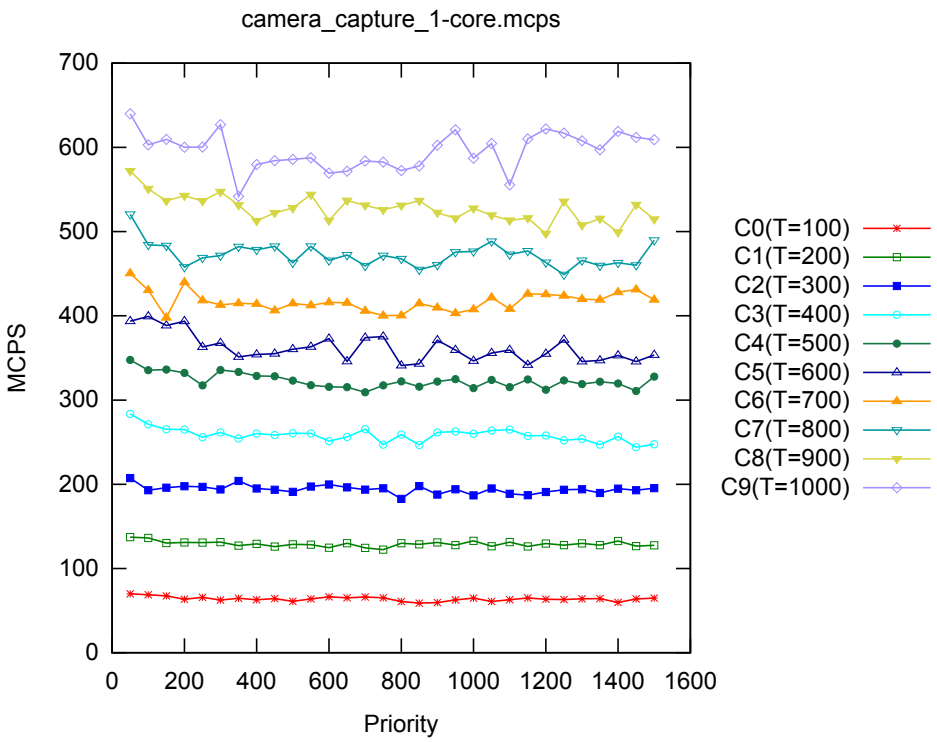


Figura 62: Desempenho em MCPS de Camera Capture

**Análise Experimental** Analisando o desempenho sob a métrica MCPS, os resultados podem ser visualizados na figura 62, onde pode ser observado um

pico de desempenho de cerca de 600 MCPS, observando uma parametrização de ajuste de 10 curvas C0 até C9, com valores de 100 até 1.000, com passos de tamanho 100 e priorização no intervalo de 50 até 1.500, com passos de tamanho 50. Com estes resultados e parâmetros, a melhoria de desempenho relativo foi de cerca de 234 vezes, observando que o ISS obteve um desempenho de 2,56 MCPS.

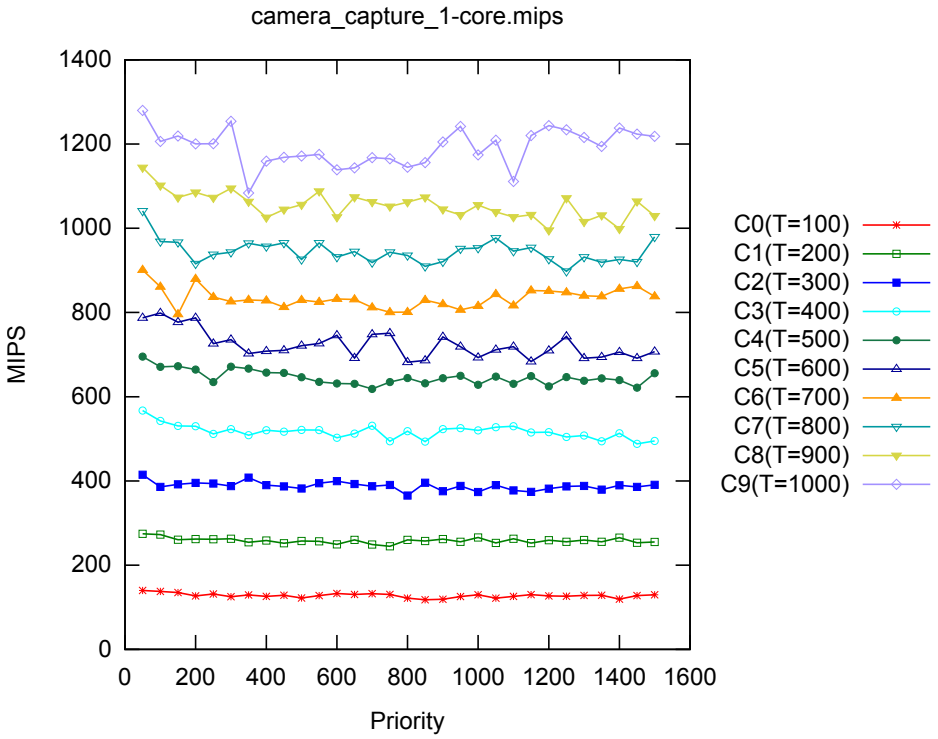


Figura 63: Desempenho em MIPS de Camera Capture

Seguindo a mesma parametrização já definida, a figura 63 exibe os resultados de desempenho na unidade de MIPS e pode ser visto um pico de cerca de 1.300 MIPS no desempenho do modelo. Já o ISS obteve um desempenho de 0,68 MIPS, o que nesta configuração representa uma melhoria do desempenho de cerca de 1.912 vezes do modelo proposto sobre o ISS.

Na figura 64, as curvas de tempo simulado geradas são exibidas no gráfico, permitindo confirmar o modelo teórico definido anteriormente. Com uma parametrização de ajuste de 21 e de priorização de 1.456, foi gerado um tempo simulado de 2,92271101776e+12 picosegundos com 6 simulações, com erro de precisão de 0,24% relativo ao ISS que obteve 2,92975264987e+12 picosegundos e desvio padrão de 2,13385773e+10 picosegundos (±0,73%). Foi obtido pelo modelo proposto um desempenho de 13,33 MCPS e 26,66 MIPS que representa uma melhoria de desempenho de cerca de 5 e 39 vezes, respectiva-

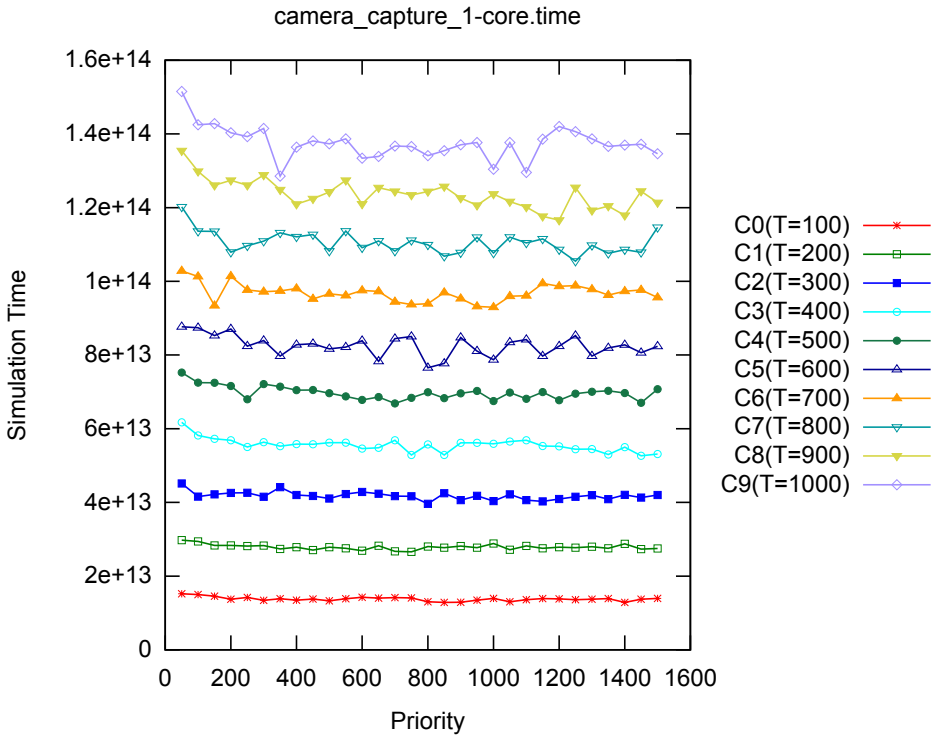


Figura 64: Tempo simulado de Camera Capture

mente, quando comparado ao ISS que obteve 2,56 MCPS e 0,68 MIPS.

Com estes resultados experimentais fica evidente o determinismo temporal do modelo, permitindo que resultados possam ser estimados sem necessidade de simulações e com isto proporcionando uma ferramenta confiável de avaliação e desenvolvimento em plataformas virtuais. A principal característica desta aplicação é sua natureza totalmente dedicada a captura e exibição de imagens, sem qualquer processamento de software relevante, gerando uma taxa de cerca de 3,41 quadros por segundo e tráfego total de cerca de 54 MB no barramento. Isto permite uma visão clara do comportamento sob um ambiente de simulação onde praticamente todo o tempo de simulação é consumido acessando e controlando dispositivos da plataforma, usando massivamente os recursos de comunicação externa da plataforma.

**Conclusões** Quando está sendo realizado o desenvolvimento de sistemas para captura e exibição de imagens, um dos pontos mais críticos é a verificação das taxas de quadros por segundo atingidas pelo sistema. Geralmente este valor de quadros é estabelecido como um requisito não funcional do projeto e é uma restrição que deve ser atendida pelo sistema. Neste cenário, o modelo proposto é capaz de oferecer diferentes taxas de execução através

da modificação do parâmetro de ajuste, permitindo que mais ou menos operações executem em um determinado espaço de tempo simulado.

Caso se deseje duplicar a taxa de execução do processador, é necessário que o valor do parâmetro de ajuste seja reduzido pela metade e a prioridade seja mantida. Com ajuste de valor 11 e prioridade de valor 1.456, a simulação atinge um desempenho de cerca de 5,27 quadros por segundo. Este resultado é interessante e consegue fornecer ao projetista uma noção real sobre o funcionamento do sistema, visto que dobrando a taxa de execução do processador foi atingido um valor menor que o dobro da taxa de quadros esperada. Analisando os resultados, é possível identificar se existem gargalos na comunicação ou se trata de limitações inerentes do próprio dispositivo.

Desta maneira, o projetista do sistema é capaz de avaliar o sistema como se ele estivesse executando em diferentes arquiteturas, com diferentes taxas de acesso ao barramento e gerando diferentes taxas de quadros por segundo. Com esta capacidade, é permitida a avaliação rápida de diversas arquiteturas de sistemas, combinando diferentes taxas de execução de processamento e de acesso ao barramento de comunicação.

#### 6.4.1.2 Device Control

**Descrição** Esta aplicação foi desenvolvida como prova de conceito para verificar as funções de acesso e controle por software. Sua função é realizar operações de inicialização e ajuste do sistema de interrupção, através do uso das capacidades de modelagem propostas. Uma vez realizadas as operações de inicialização e configuração, o módulo de Timer possui duas instâncias que geram interrupções em intervalos de 10 e 50 microssegundos, podendo acontecer ao mesmo tempo.

**Análise Experimental** Na figura 65, são obtidas informações sobre o desempenho em MCPS com parametrização de prioridade e de ajuste. Os valores de priorização estão definidos no eixo horizontal, variando de 50 até 1.500, com passos de tamanho 50, e os de ajuste são referentes as 10 curvas denotadas por C0 até C9, com valores no eixo vertical, variando de 100 até 1.000, com passos de tamanho 100. No ISS foi obtido desempenho de 0,35 MCPS,

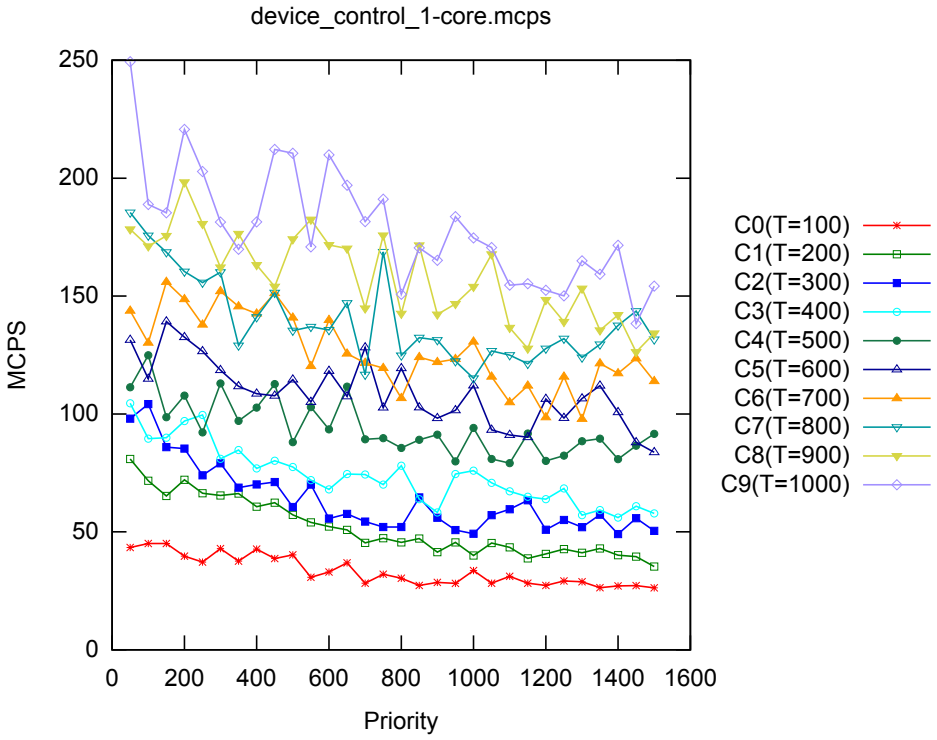


Figura 65: Desempenho em MCPS de Device Control

enquanto que no modelo proposto foi observado um pico de cerca de 250 MCPS, ou seja, uma melhoria de 714 vezes.

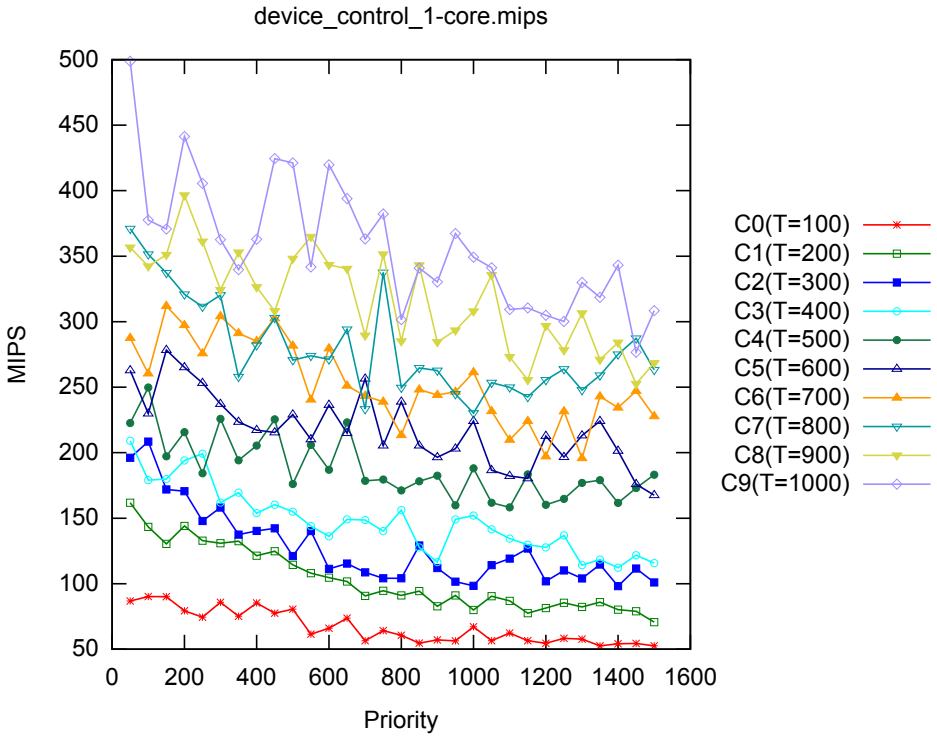


Figura 66: Desempenho em MIPS de Device Control

Na análise de desempenho em MIPS da abordagem proposta, vista na fi-

gura 66, são feitas diversas simulações com diferentes valores de priorização e de ajuste, conforme definição já realizada no parágrafo anterior. Foi obtido um pico de desempenho de cerca de 500 MIPS pelo trabalho proposto, enquanto que no resultado do ISS foi obtido um desempenho de 0,22 MIPS, gerando uma melhoria de cerca de 2.273 vezes no desempenho.

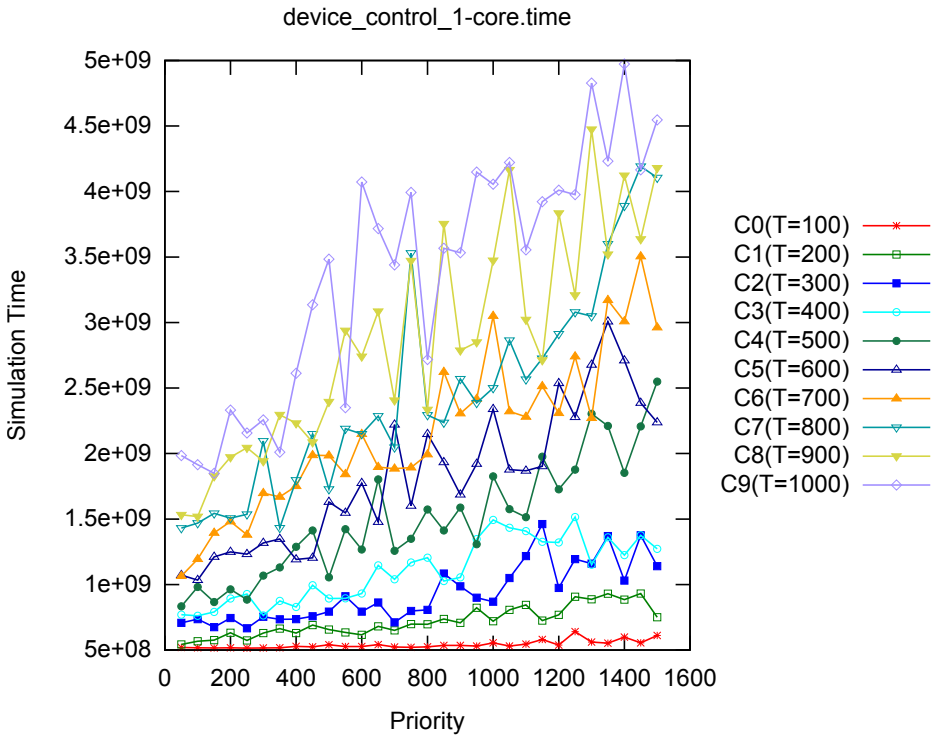


Figura 67: Tempo simulado de Device Control

Focando no tempo simulado, ilustrado na figura 67, é possível observar que a mesma parametrização de priorização e de ajuste geram gráficos bem distintos de tempo. Apesar de sua aparência imprecisa e ruidosa, para valores de ajuste maiores, este efeito é devido a alta resolução de tempo em picosegundos observado neste gráfico. No modelo ISS o tempo simulado é de 1,029915e+9 picosegundos.

O leitor crítico pode imaginar que esta aplicação representa um contra exemplo do comportamento temporal previsto anteriormente, mas este caso reflete exatamente o mesmo comportamento, só que em diferentes granularidades. Quanto maior a priorização, maior o número de blocos básicos executados e consequentemente maior a granularidade temporal, causando níveis de erros superiores. Esta aplicação possui um comportamento de tempo constante pelo fato de seu comportamento estar diretamente atrelado ao tempo da plataforma, ou seja, sua execução é finalizada quando um determinado

tempo for contabilizado.

Utilizando parâmetros de ajuste com valor 339 e priorização com valor 1.013 em 794 simulações, é obtido um tempo simulado de 1,062372203e+9 picosegundos que representa um erro de 3,15% com relação ao ISS e desvio padrão de 1,18398276e+8 picosegundos (±11,14%). Com relação ao desempenho, foi obtido pelo modelo proposto 56,94 MCPS e 113,89 MIPS que aumentam em 163 e 518 vezes o desempenho atingido pelo ISS que foi de 0,35 MCPS e 0,22 MIPS.

**Conclusões** O objetivo principal deste estudo de caso é ilustrar o comportamento do modelo proposto quando o tempo simulado também é parte da funcionalidade da aplicação. Neste cenário, o modelo teórico de tempo assume um papel de granularidade das operações realizadas, permitindo que um número diferente de eventos sejam agrupados em uma determinada unidade de tempo. Devido a sua configurabilidade, diferentes parâmetros podem ser utilizados para que diversos níveis de precisão e granularidade possam ser explorados pelo projetista do sistema, avaliando inclusive se está ocorrendo a perda de eventos gerados pela plataforma.

#### 6.4.1.3 Hello World

**Descrição** É uma das mais simples aplicações possíveis de ser feita, o programa Hello World consiste simplesmente em imprimir na tela a mensagem "Hello World!". A ideia é enfatizar e ilustrar os problemas de estimativa de tempo inerentes da simulação nativa e como os algoritmos utilizados são eficientes em contextos favoráveis. É importante ressaltar que nesta aplicação existe somente um único bloco básico, que é a própria aplicação, o que torna o parâmetro de prioridade sem efeito, pois não existem blocos a serem priorizados.

**Análise Experimental** Na figura 68, pode ser visto o desempenho da aplicação em termos de MCPS, no intervalo de prioridade de 50 até 1500, com passos de tamanho 50, e 10 curvas de ajuste com nomes C0 até C9, com valores de 100 até 1000, com intervalos de tamanho 100. No ISS foi obtido um

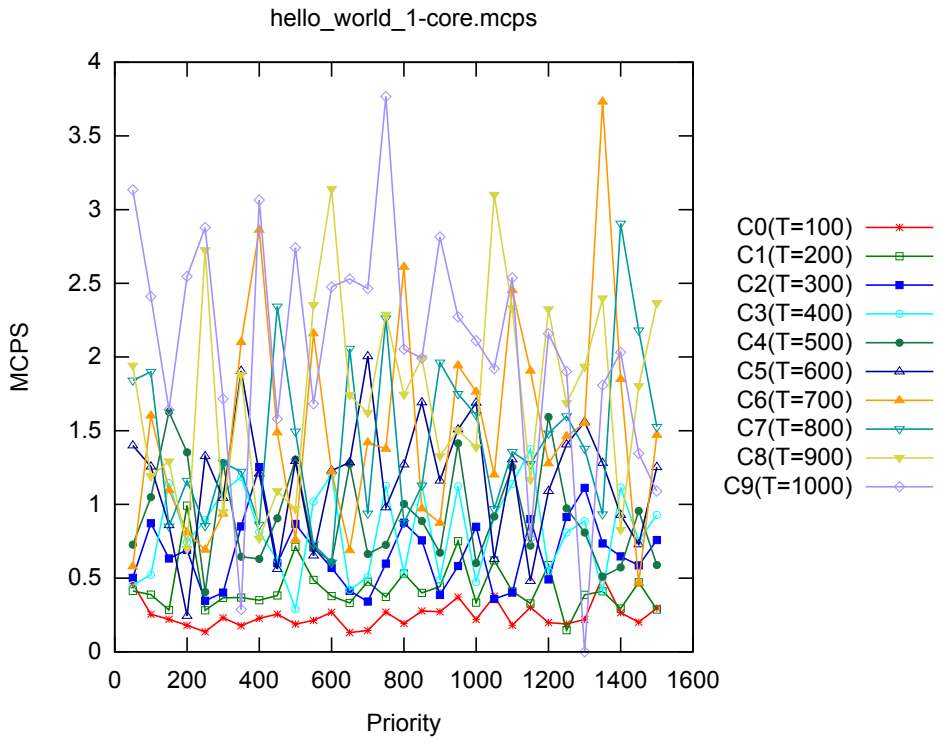


Figura 68: Desempenho em MCPS de Hello World

desempenho muito pequeno de 0,0012 MCPS, enquanto que no modelo proposto foi obtido um pico de 3,5 MCPS, ou seja, um aumento de 2.917 vezes no desempenho obtido.

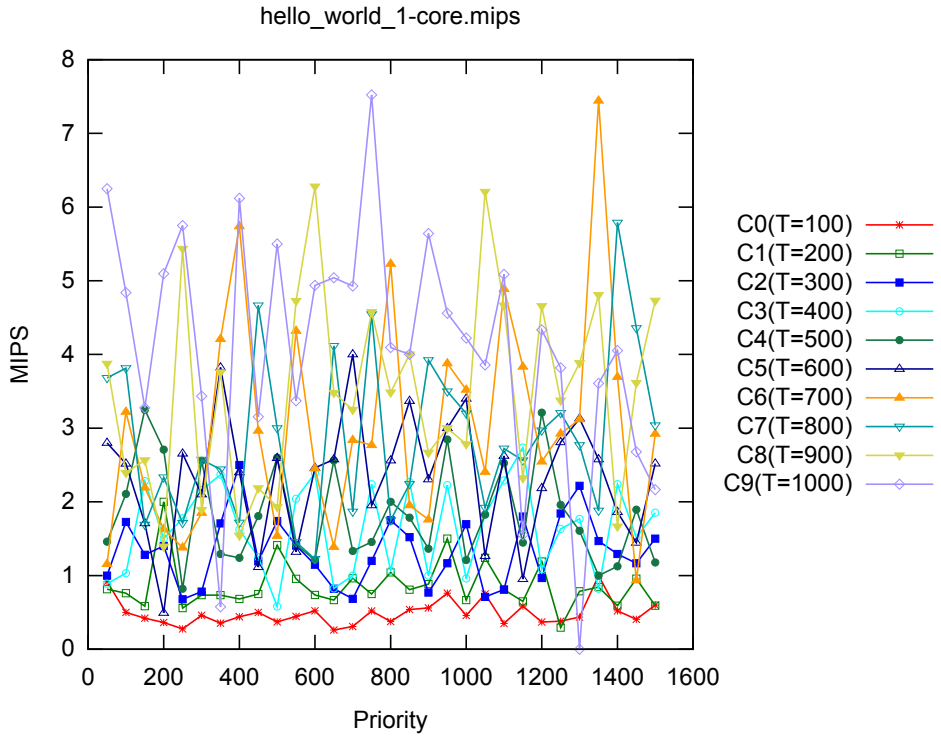


Figura 69: Desempenho em MIPS de Hello World

No modelo ISS, no relatório de simulação da execução foi obtido um tempo simulado de 2,383e+6 picosegundos com desempenho de 0,0005 MIPS, confirmando os argumentos expostos anteriormente. No modelo proposto foi atingido um pico 7 MIPS e ganho de desempenho de cerca de 14.000 vezes, com a mesma parametrização já descrita anteriormente, como pode ser visto na figura 69 que segue a mesma parametrização e intervalos definidos anteriormente.

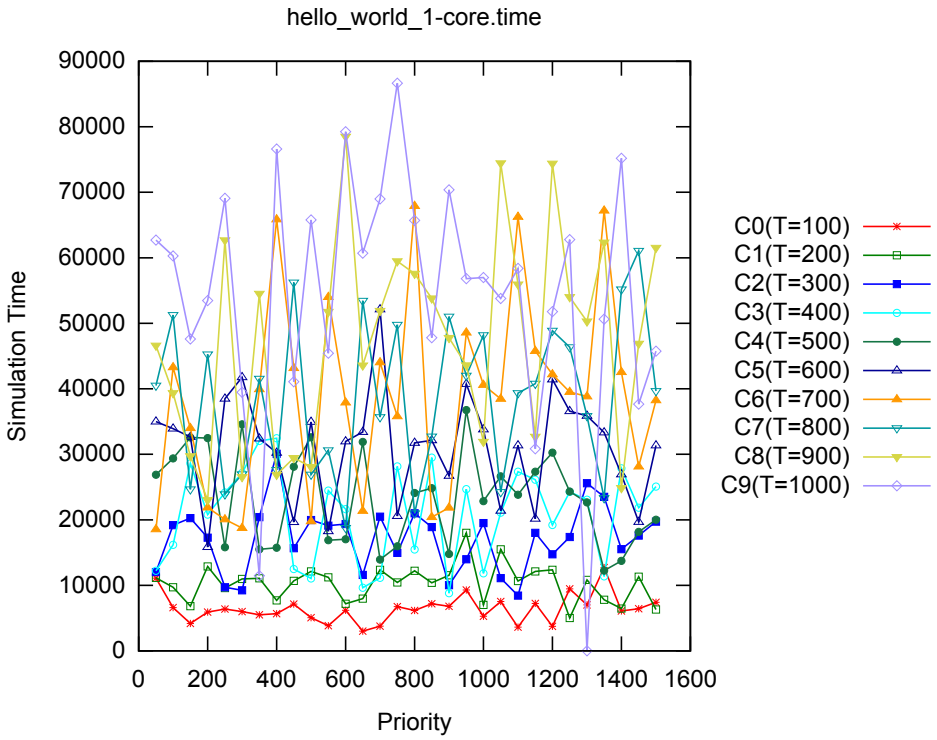


Figura 70: Tempo simulado de Hello World

Quando se observa, na figura 70, o comportamento das curvas de ajuste, é visualizado um comportamento ruidoso e até mesmo aleatório. Como só existe uma única amostragem de tempo, decorrente do único bloco básico, a estimativa de tempo e as técnicas de filtragem de tempo se tornam ineficazes por só ter uma única amostra disponível. Esta amostra é obtida através do tempo real de execução da máquina nativa, sendo muito susceptível à variâncias e ao não determinismo. Este exemplo ilustra como as técnicas propostas são eficientes em contextos mais favoráveis, usando aplicações mais realistas e complexas.

Para igualar o comportamento da simulação ISS, são utilizados os parâmetros 52.972 de ajuste e 853 de prioridade, gerando uma estimativa de tempo simulado de 2,030897e+6 picosegundos em 2.582 simulações, com desempe-

nho de 88,61 MCPS e 177,22 MIPS que representa uma melhoria de 73.842 e 354.440 vezes no desempenho, respectivamente. Com esta parametrização e considerando uma amostra de tempo média, uma estimativa com erro de 14,77% em comparação ao resultado gerado pelo ISS e desvio padrão de 1,824468e+6 picosegundos (±89,83%).

**Conclusões** A aplicação Hello World foi adotada neste estudo de caso para ilustrar a importância do cálculo da média de execução pelo núcleo de simulação, uma vez que os resultados vistos sofrem fortes influências do não determinismo da execução na máquina nativa. Este programa funciona como um caso base dos experimentos, deixando evidente o pior cenário possível para realização das estimativas de tempo simulado.

#### 6.4.1.4 LCD Pattern

**Descrição** Para demonstrar o acesso massivo em um dispositivo da plataforma, foi desenvolvida uma aplicação para exibir padrões de imagens em um componente LCD da plataforma. Este módulo é o Sharp LQ043T3DX02 com resolução de 480 colunas por 272 linhas, com interface RGB serial e 24 bits de definição de cores, com uma frequência de operação de 9 MHz. São exibidos 10 quadros de imagem, sendo os 5 primeiros mostrando um padrão de 3 partes com as cores vermelha, verde e azul em degradê, como forma verificar a integridade da imagem, e os 5 quadros seguintes são imagens de fractais do Triângulo de Sierpinski com iterações de 5.000 até 10.000, com passos de tamanho 1.000.

**Análise Experimental** Na primeira análise dos resultados é verificado o desempenho em MCPS do modelo proposto, observando uma parametrização de 50 até 1.500, com intervalos de tamanho 50 e 10 curvas de ajuste C0 até C9, com valores que variam de 100 até 1.000, com espaçamento de tamanho 100. Na figura 71, é feita a ilustração destes resultados e pode ser observado o efeito do não determinismo da simulação nativa. Entretanto, é observável um padrão no nível horizontal para cada uma das 10 curvas, onde todos os parâmetros estão concentrados em um desempenho uniforme.

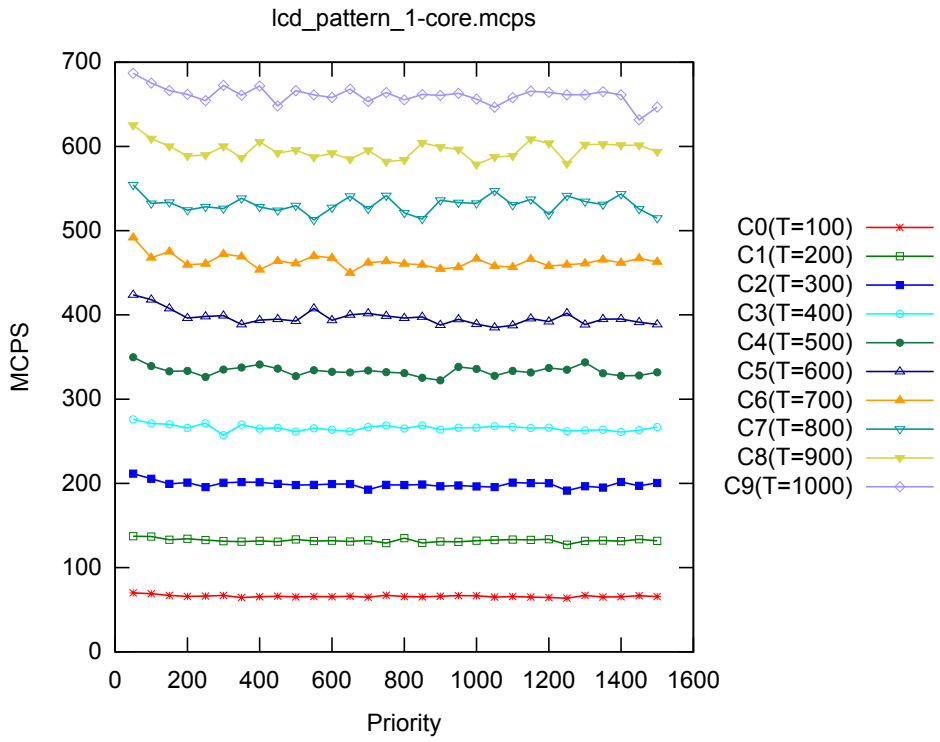


Figura 71: Desempenho em MCPS de LCD Pattern

Apresentando um pico de desempenho de cerca de 700 MCPS, o modelo proposto oferece uma desempenho de cerca de 99 vezes superior ao obtido pelo ISS, com 7,11 MCPS, considerando esta parametrização. Esta baixa melhoria de desempenho pode ser atribuída a baixa velocidade do dispositivo LCD, que opera em 9 MHz, enquanto o processador trabalha em uma taxa de 1 GHz. Neste cenário, grande parte do tempo de execução é consumido aguardando que a escrita no dispositivo seja concluída e não efetivamente executando o software.

Na figura 72, o padrão de desempenho na unidade de MIPS é exibido, considerando a mesma parametrização descrita anteriormente e os mesmos intervalos. É interessante notar que este desempenho reflete somente a estimativa de execução do software, excluindo todo o tempo consumido em operações de entrada e saída no barramento. No modelo ISS foi atingido um desempenho de 0,61 MIPS, enquanto que no modelo proposto foi obtido um pico de cerca de 1.400 MIPS, representando uma melhoria de cerca de 2.295 vezes no desempenho.

Adotando a mesma parametrização já definida, na figura 73 é feita a análise das estimativas de tempo geradas em uma escala de picosegundos. É interessante observar que o tempo previsto pelo modelo teórico é exibido neste

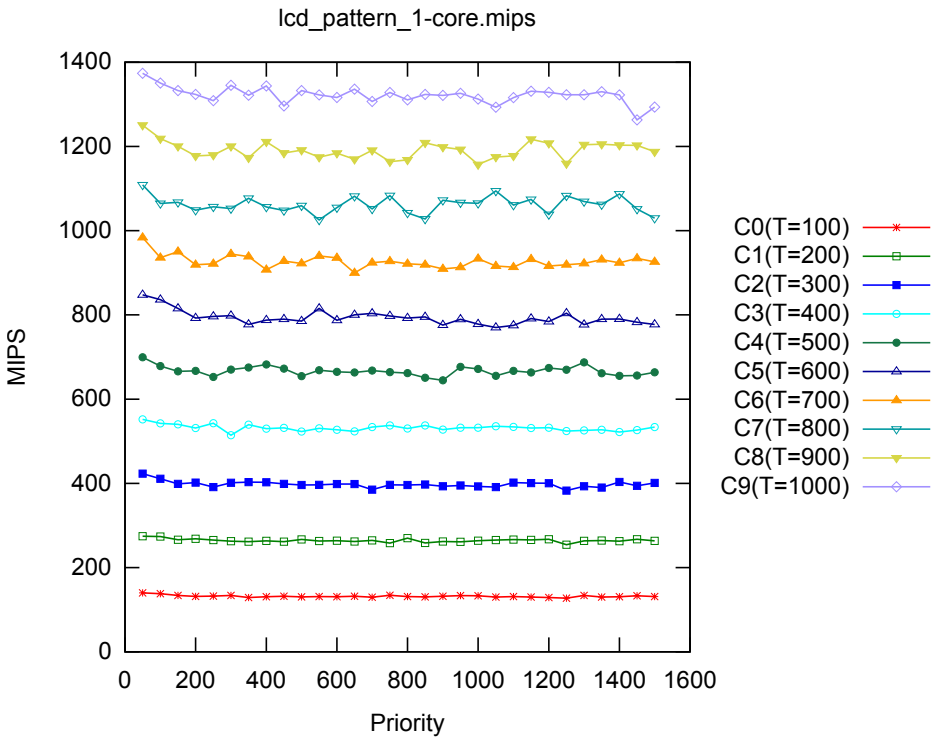


Figura 72: Desempenho em MIPS de LCD Pattern

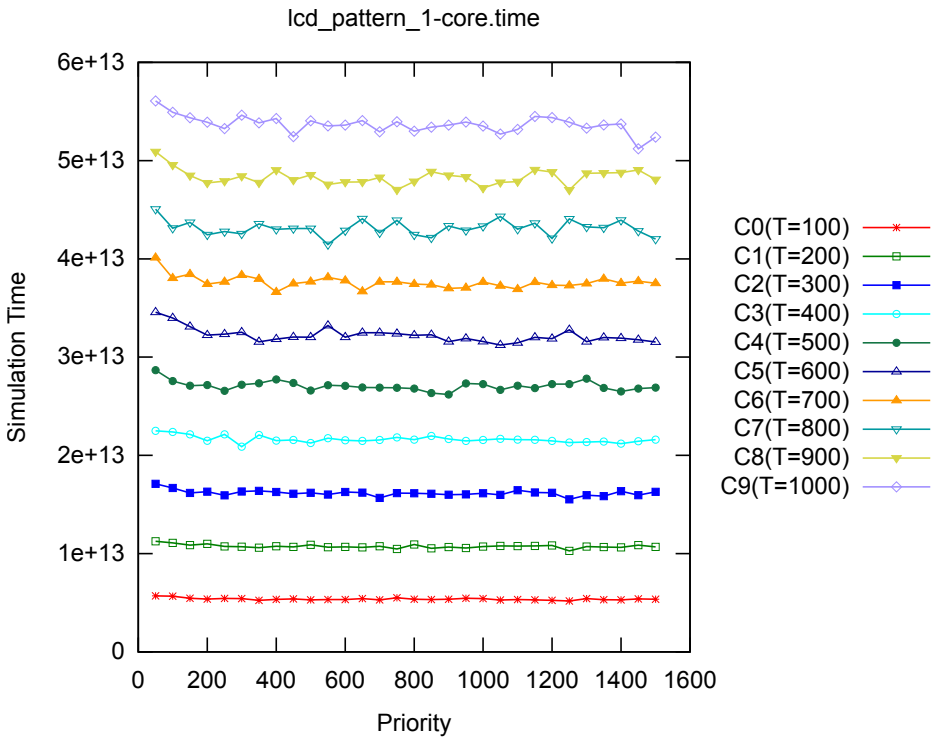


Figura 73: Tempo simulado de LCD Pattern

padrão de curvas, desta forma é possível, com algumas poucas simulações, predizer o comportamento temporal da aplicação no modelo proposto. Este recurso se mostra muito interessante, principalmente em simulações extrema-

mente longas e complexas, reduzindo drasticamente o número de simulações que são necessárias.

No ISS, o tempo simulado obtido foi de 2,01678625285e+12 picosegundos, com cerca de 20 MB de dados transferidos em acessos alinhados ao dispositivo (5 acessos de 32 bits por pixel) e uma taxa média de 4,96 quadros por segundo. Para equiparar este comportamento, foi utilizada uma parametrização de 42 de ajuste e de 1.050 para prioridade em 8 simulações, gerando um tempo estimado de 2,029956668087e+12 picosegundos, que representa um erro de cerca de 0,65% e desvio padrão de 4,1518777097e+10 picosegundos (±2,04%). Foi atingido um desempenho de 24,64 MCPS e 49,27 MIPS, o que representa uma melhoria de aproximadamente 4 e 81 vezes, respectivamente, quando comparado ao ISS que obteve 7,11 MCPS e 0,61 MIPS.

Existe um limite temporal imposto pelo dispositivo, que foi modelado na plataforma, mas com o ajuste maior utilizado, o tempo de execução passa a ser relevante, assim como ocorre no ISS e permite uma melhoria nas taxas de desempenho geradas. No modelo proposto foi atingido uma taxa de 4,93 quadros por segundo que é um valor muito próximo ao observado no ISS que foi de 4,96 quadros por segundo, retomando mais uma vez a precisão do modelo com relação ao comportamento exibido.

**Conclusões** No desenvolvimento de sistemas com forte interação com dispositivos, como acontece no caso da aplicação LCD Pattern, é interessante que o projetista seja capaz de avaliar se os componentes de software desenvolvidos são capazes de atender as restrições impostas. No estudo de caso foi feita a configuração do modelo proposto para operar com uma taxa em torno de 5 quadros por segundo que é o comportamento observado na plataforma virtual de referência, com erro médio nas estimativas em torno de 6,39%.

Devido a sua configurabilidade, o modelo proposto pode ser ajustado para gerar uma taxa de quadros diferente e rapidamente permitir ao projetista verificar o comportamento do sistema. Caso se deseje dobrar a taxa de quadros, basta que o parâmetro de ajuste seja reduzido pela metade, duplicando o número de operações realizadas em uma mesma unidade com relação a plataforma de referência. Com ajuste de 19 e prioridade de 222, o modelo proposto atinge uma taxa de cerca de 6,77 quadros segundo, indicando que mesmo

com o dobro de velocidade, existem gargalos no acesso ao dispositivo ou que o mesmo possui uma taxa limitada de dados. Este tipo de análise é extremamente útil para que o projetista desenvolva a arquitetura mais adequada para atender seus requisitos, validando através de simulações as decisões tomadas e reduzindo as chances de que os componentes de software desenvolvidos apresentem problemas.

#### 6.4.1.5 Mergesort

**Descrição** Aplicando um algoritmo clássico de ordenação com Mergesort (63), este estudo de caso tira proveito do projeto do algoritmo completamente adequado para ordenar grandes conjuntos de dados e fazer estas operações utilizando mídias externas de dados. Nesta aplicação o algoritmo Mergesort é utilizado para realizar a ordenação de um conjunto de dados armazenados no componente de armazenamento da plataforma Micron MT28F320J3 (Storage), através do uso de ponteiros de programação, utilizando-o tanto para aquisição dos dados como para escrita do resultado gerado.

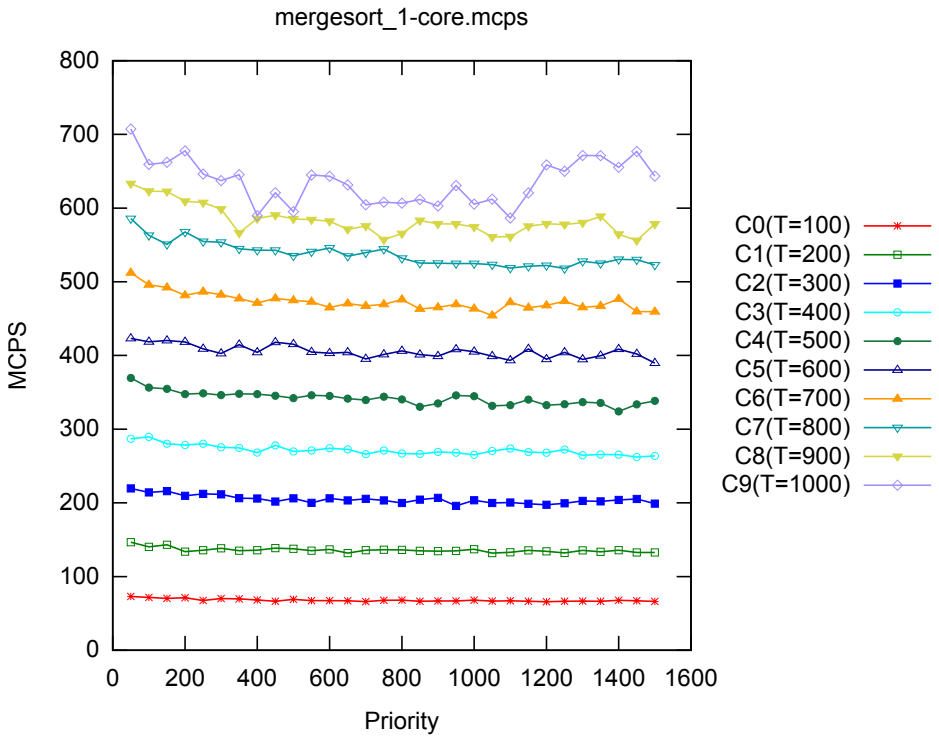


Figura 74: Desempenho em MCPS de Mergesort

**Análise Experimental** A primeira métrica a ser analisada é o desempenho em MCPS, considerando a configuração de ajuste variando de 100 até 1.000, com intervalos de tamanho 100, e de priorização de 50 até 1.500, com passos de tamanho 50. Na figura 74 é feita a exibição do desempenho em MCPS obtido pelo modelo proposto, considerando a configuração de ajuste e priorização descrita.

Pode ser observado um claro padrão de comportamento nas curvas do gráfico, o que é decorrente do modelo teórico previsto para o comportamento de tempo simulado, com um pico de desempenho de cerca de 700 MCPS. Este desempenho representa uma melhoria de aproximadamente 342 vezes quando comparado ao ISS que obteve 2,05 MCPS de desempenho.

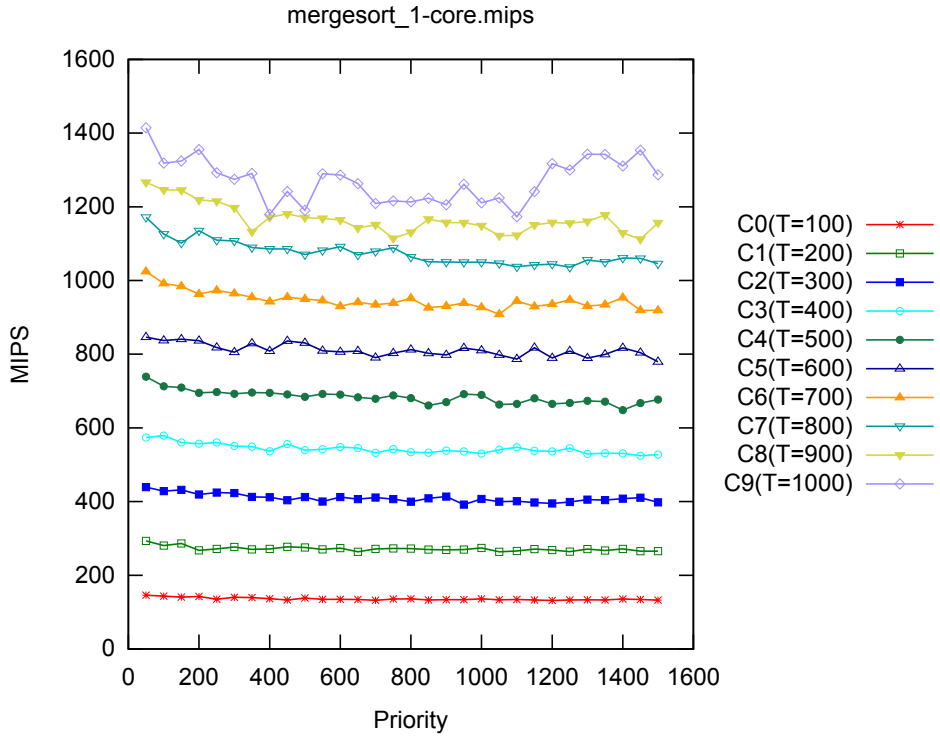


Figura 75: Desempenho em MIPS de Mergesort

Na figura 75, seguindo a mesma parametrização descrita anteriormente para ajuste e priorização, é exibido o desempenho do modelo para a aplicação Mergesort utilizando a métrica MIPS. Foi atingido um pico de desempenho de cerca de 1.400 MIPS, enquanto que no modelo ISS foi obtido um desempenho de 0,65 MIPS, assim o modelo proposto proporcionou um ganho de desempenho de aproximadamente 2.154 vezes sobre o ISS.

Observando a figura 76 pode ser visto os tempos simulados estimados pelo modelo proposto para a aplicação Mergesort, correspondendo exatamente

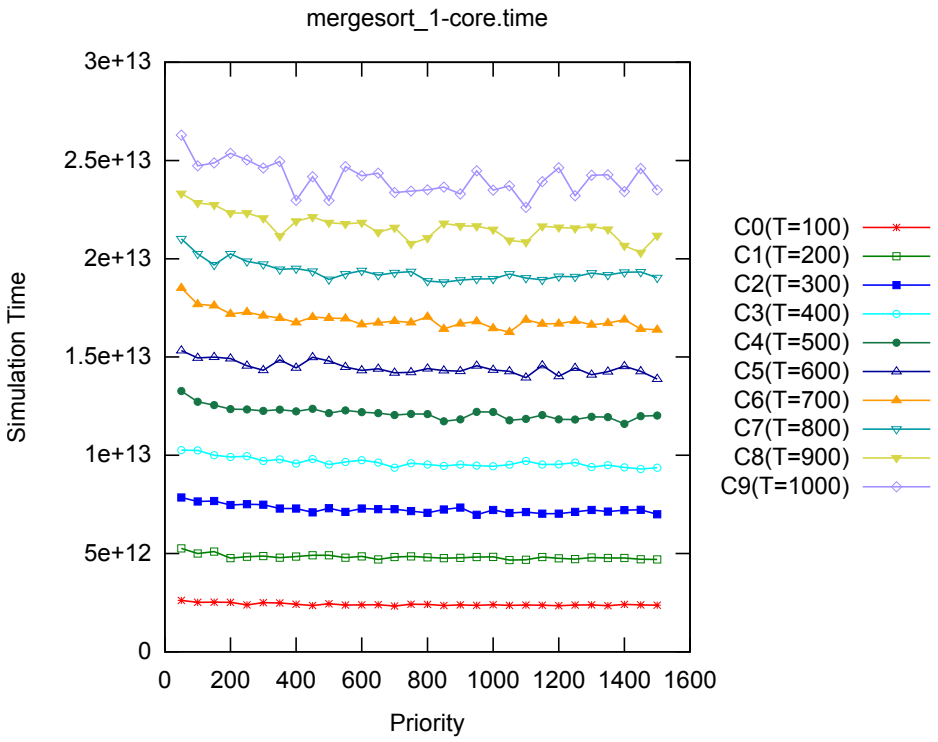


Figura 76: Tempo simulado de Mergesort

as previsões do modelo teórico estabelecido na estimativa de tempo simulado. Com esta regularidade, é possível ao projetista do sistema obter informações sobre a execução da simulação sem ter que efetivamente realizar as simulações, necessitando somente que as constantes associadas às curvas sejam calculadas.

Na simulação utilizando ISS foi obtido um tempo simulado de 5,61208925e+11 picosegundos, tempo consumido com a execução do algoritmo e os constantes acessos ao dispositivo de armazenamento de dados do barramento. Para que um comportamento equivalente fosse observado no modelo proposto, foi necessário definir parâmetros de ajuste com valor 26 e de priorização com valor 644, o que resultou em uma estimativa de tempo simulado de 5,67610364106e+11 picosegundos em 9 simulações, com erro associado de 1,14% e desvio padrão de 5,541127707e+9 picosegundos (±0,97%), quando comparado ao ISS.

Os desempenhos obtidos foram de 15,65 MCPS e 31,29 MIPS que representam uma melhoria de cerca de 8 e 48 vezes, respectivamente, quando comparados aos desempenhos obtidos pelo modelo ISS foram de 2,05 MCPS e 0,65 MIPS. Apesar do acesso intensivo a unidade de armazenamento externa e das limitações decorrentes, o modelo proposto conseguir proporcionar uma me-

lhoria significativa de desempenho em ambas as métricas, com baixo nível de erro na estimativa de tempo simulado.

**Conclusões** A aplicação Mergesort é um algoritmo para ordenação com complexidade $\Theta(n \log n)$ e esta característica indica que o número de operações realizadas só depende do tamanho do vetor de entrada que será ordenado. Como o tamanho do conjunto de dados a ser ordenado é fixo e os dados estão armazenados em uma unidade de memória flash, este estudo de caso se destina basicamente a mostrar o comportamento do Mergesort em um contexto de entrada e saída intensivo.

Os resultados observados sempre se caracterizam por uma análise comparativa entre dois modelos de processamento, sem estabelecer um índice ou unidade que permita uma comparação entre diferentes plataformas. Entretanto, informações referentes aos números de transações de entrada e saída realizadas dão uma exata contagem da demanda da aplicação pelos dados durante a execução do algoritmo. Neste estudo de caso foram realizados 2.964.350 acessos de leitura no barramento, solicitando dados de 32 bits de largura, e 983.043 operações de escrita com 32 bits de largura. Estas informações são essenciais para o melhor projeto do sistema de interconexão da plataforma, uma vez que consegue exibir para o projetista qual é a demanda exata de acesso da aplicação. Utilizando o parâmetro de ajuste, o projetista é capaz de modificar a taxa de execução do processador, criando situações com mais ou menos tráfego no barramento do sistema.

#### 6.4.1.6 Análise Comparativa

Esta subseção é dedicada a resumir e comentar com uma visão geral os resultados obtidos pelas aplicações de entrada e saída intensiva, focando nos aspectos de desempenho do modelo proposto (HdSC) e do modelo clássico ISS, na melhoria de desempenho obtida e no erro percentual das estimativas geradas. Em todas estas métricas, os resultados gerados pelo ISS são considerados como referência para o cálculo da melhoria de desempenho e do erro relativo do tempo simulado.

Na tabela 14, a primeira coluna diz respeito ao nome da aplicação em

Tabela 14: Comparativo de Aplicações de Entrada e Saída Intensiva

| Aplicação | HdSC | ISS | Desempenho | Erro |
|---|---|---|---|---|
| Camera Capture | 13,33 MCPS<br>26,66 MIPS | 2,56 MCPS<br>0,68 MIPS | 6x<br>42x | 0,24% |
| Device Control | 56,94 MCPS<br>113,89 MIPS | 0,35 MCPS<br>0,22 MIPS | 178x<br>567x | 3,15% |
| Hello World | 88,61 MCPS<br>177,22 MIPS | 0,0012 MCPS<br>0,0005 MIPS | 92.808x<br>445.480x | 14,77% |
| LCD Pattern | 24,64 MCPS<br>49,27 MIPS | 7,11 MCPS<br>0,61 MIPS | 4x<br>81x | 0,65% |
| Mergesort | 15,65 MCPS<br>31,29 MIPS | 2,05 MCPS<br>0,65 MIPS | 8x<br>48x | 1,14% |

questão com seu tamanho de entrada entre parênteses, quando for aplicável. Nas demais colunas, todas as métricas consideradas são listadas em termos do desempenho do modelo proposto HdSC e do ISS, nas unidades de MCPS e MIPS, na melhoria de desempenho obtida pelo modelo proposto sobre o ISS e o erro relativo das estimativas de tempo do modelo proposto com relação ao ISS.

Como resultado final, calculando-se uma média de todas as métricas, o desempenho médio do modelo proposto foi de cerca de 39,83 MCPS e 79,67 MIPS e o desempenho do ISS foi de cerca de 2,41 MCPS e 0,43 MIPS, com melhoria média de cerca de 17 e 185 vezes, respectivamente. O erro médio foi de aproximadamente 3,99%, considerando as parametrizações de ajuste e prioridade definidas nos exemplos listados.

### 6.4.2 Computação Intensiva

Nesta subseção, as aplicações de computação intensiva são listadas, fornecendo detalhes sobre seu funcionamento e como fazem uso intensivo dos recursos computacionais disponíveis na plataforma. As aplicações utilizadas nos experimentos realizados procuram contemplar os diferentes aspectos desta classe de aplicações, sendo listadas a seguir:

- CoreMark: aplicação de referência desenvolvida para medir o desempenho de processadores, sendo implementada na linguagem C e utilizando implementações de algoritmos não sintéticos para processamento de listas, manipulação de matrizes, simulação de máquina de estados e

cálculo de CRC. Por utilizar algoritmos de ampla utilização e ter o objetivo de se tornar um padrão da indústria para avaliação do desempenho de processadores, esta aplicação de referência vai permitir uma comparação quantitativa do desempenho de execução da abordagem proposta com outros sistemas que também utilizam esta medição;

- Dhrystone: é uma das aplicações de referência mais conhecidas para medição de desempenho de processadores, considerando a aritmética inteira em sua execução. Os algoritmos utilizados são sintéticos, ou seja, foram especificamente criados para realizar um conjunto de operações que exercitam determinadas funcionalidades do processador. A medição realizada por esta aplicação permite estabelecer uma relação com outras plataformas e permitir a comparação quantitativa do desempenho obtido pela abordagem proposta;

- MiBench: este pacote de aplicações possui um conjunto de aplicações e algoritmos relevantes em diversas áreas da indústria como a automotiva, de consumo, de rede de computadores, de segurança e de telecomunicações. O principal objetivo da avaliação destas aplicações na abordagem proposta é demonstrar sua utilização em cenários diversificados, com algoritmos realistas que permitem avaliar as funcionalidades propostas através dos estudos de caso.

Nas próximas subseções serão fornecidos maiores detalhes sobre a descrição de cada aplicação, além de detalhes sobre como a análise experimental foi conduzida e seus resultados, com exceção do conjunto de aplicações MiBench que estão agrupadas no anexo B deste documento, para melhorar o entendimento do leitor. No final de cada subseção serão realizadas conclusões sobre o estudo de caso realizado, contextualizando no estado da arte considerado e com cenários distintos de utilização, fazendo referência para sistemas similares.

#### 6.4.2.1 CoreMark

**Descrição** Devido a necessidade de uma referência para analisar o desempenho de sistemas, a aplicação de referência CoreMark foi desenvolvida pela

Embedded Microprocessor Benchmark Consortium (EEMBC) (74) para avaliação de desempenho. Esta aplicação tem o objetivo de servir como uma referência de desempenho para processadores, gerando um resultado numérico mensurável e comparável de desempenho. Com este resultado é possível analisar e comparar diferentes arquiteturas de processadores, com diferentes conjuntos de instruções, de forma independente da tecnologia utilizada.

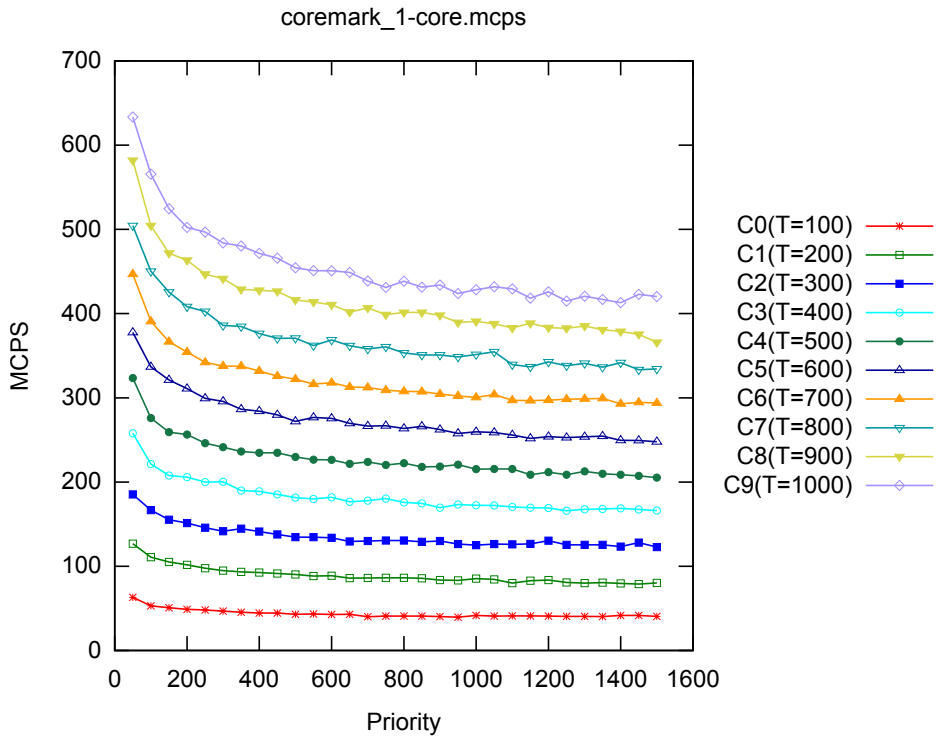


Figura 77: Desempenho em MCPS de CoreMark

**Análise Experimental** Na análise de desempenho em MCPS, ilustrado na figura 77, a aplicação CoreMark é simulada com diferentes valores de priorização, com valores de 50 até 1.500, com passos de tamanho 50, e de ajuste, no intervalo de 100 a 1.000, com passos de tamanho 100. Nesta parametrização pode ser observado um pico de desempenho de 600 MCPS, que representa uma melhoria de 561 vezes sobre o modelo ISS, que apresenta desempenho de 1,07 MCPS.

Na figura 78, é ilustrado o desempenho em MIPS da aplicação CoreMark no modelo proposto, usando a mesma parametrização descrita anteriormente. O resultado observado no gráfico demonstra a escalabilidade dos parâmetros, além das altas taxas de desempenho com um pico de cerca de 1.200 MIPS que podem ser atingidas pelo modelo proposto. Neste intervalo de prio-

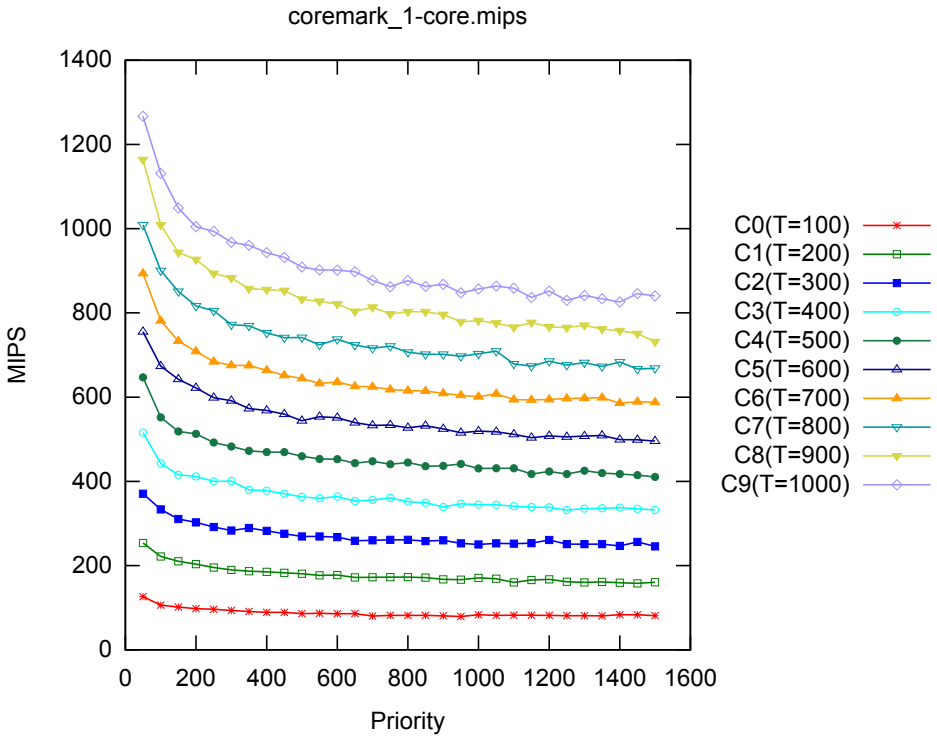


Figura 78: Desempenho em MIPS de CoreMark

rização e de ajuste foi obtida uma melhoria máxima de desempenho de 1.846 vezes, quando comparando ao ISS que obteve 0,65 MIPS.

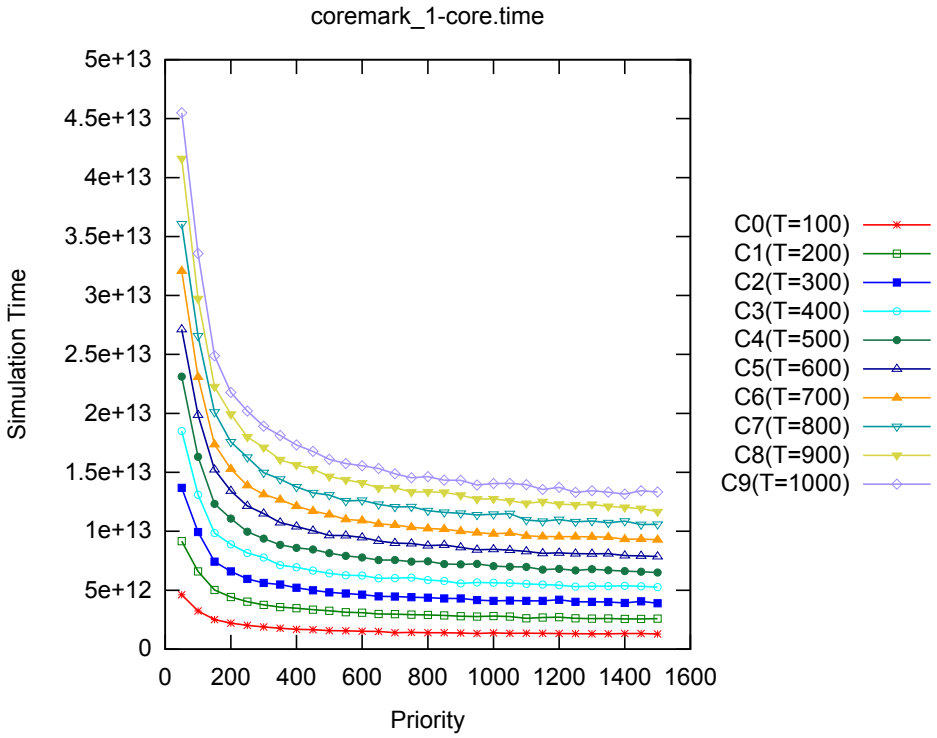


Figura 79: Tempo simulado de CoreMark

Analisando o tempo simulado, através da visualização da figura 79, que é

uma representação equivalente do número de instruções, é observado exatamente o comportamento previsto pelo modelo de estimativa de tempo proposto, com a parametrização de prioridade no eixo horizontal e de ajuste no eixo vertical com as 10 curvas C0 até C9. Como a aplicação não possui acoplamento com o tempo, sua execução e finalização são decorrentes do fim de suas iterações e conclusão de suas operações. No modelo ISS foi obtido um tempo simulado de 1,1638662324e+13 picosegundos, que deve ser aproximado o máximo possível pela definição dos parâmetros de priorização e de ajuste, maximizando o desempenho obtido e minimizando o erro gerado. Considerando o intervalo acima, foram calibrados parâmetros de ajuste de 867 e de priorização de 973, gerando uma estimativa de tempo simulado de 1,1792995136889e+13 picosegundos em 3 simulações, com erro aproximado de cerca de 1,32% e desvio padrão de 9,9811403376e+10 picosegundos (±0,84%). O desempenho atingido pelo modelo proposto foi de 362,07 MCPS e 724,13 MIPS que representa um ganho de desempenho de simulação de aproximadamente 340 e 1.113 vezes, respectivamente, quando comparado ao ISS que obteve 1,07 MCPS e 0,65 MIPS.

**Conclusões** Na plataforma de referência construída para realização dos estudos de caso a aplicação CoreMark obteve uma pontuação de aproximadamente 0,258 CoreMark/MHz, utilizando um modelo de processador baseado em ISS. A plataforma Aeroflex Gaisler LEON3 (70), que também é baseada no processador SPARC (64), obteve uma pontuação de 1,96 CoreMark/MHz (75) rodando em tecnologia FPGA Xilinx Spartan-6 (76). Executando em ambiente de plataforma virtual, a plataforma de referência baseada em ISS apresenta um desempenho cerca de 7,54 vezes menor do que a implementação em FPGA.

Para obter o mesmo comportamento da plataforma de referência, que utiliza um modelo ISS de processador, uma plataforma com o modelo proposto foi configurada neste estudo de caso com parâmetros de ajuste e prioridade com valores de 837 e 1.058, respectivamente, para que um comportamento muito próximo ao observado na plataforma de referência. Os resultados das simulações indicaram um desempenho muito próximo a plataforma baseada em ISS, de acordo com a pontuação do CoreMark, com um erro médio de 0,20% nas estimativas de desempenho realizadas pela abordagem proposta.

Configurando o modelo proposto com mesmo valor de prioridade e um ajuste com valor de 111 é equivalente a realizar o mesmo número de operações em um tempo simulado cerca de 8 vezes menor, ou seja, aumentando a taxa de execução da aplicação. Esta configuração de ajuste equipara a taxa de execução com a plataforma baseada em FPGA, sendo obtido um resultado de simulação que atinge a pontuação de 1,951 CoreMark/MHz que é muito próximo ao obtido pela implementação em hardware. A principal conclusão deste estudo de caso é que a modelagem proposta gera estimativas de execução com alto desempenho e com baixo erro de precisão, além dos resultados serem consistentes com versões mais detalhadas da plataforma, uma vez que os resultados observados estão em conformidade com modelo ISS e com a implementação em FPGA consideradas.

#### 6.4.2.2 Dhrystone

**Descrição** Provavelmente a mais emblemática aplicação de referência já desenvolvida, utilizando números inteiros, o Dhrystone (77) foi criado em 1984 por Reinhold P. Weicker com o objetivo de avaliar o desempenho de processadores. Em sua implementação são usados algoritmos sintéticos, ou seja, artificialmente criados para imitar algum comportamento ou carga de utilização do processador. Ao invés de medir o número de instruções executadas, o Dhrystone mede o número de iterações realizadas pelo processador em um intervalo de tempo. Com esta medida independente de arquitetura ou tecnologia, é derivada a métrica DMIPS que é o número de iterações realizadas por segundo, considerando uma máquina VAX 11/780 (78) como referência.

**Análise Experimental** Na figura 80, é exibido o desempenho do modelo proposto em MCPS, considerando o mesmo tipo de parametrização das aplicações anteriores, com prioridade variando de 50 até 1.500, com passos de tamanho 50 e 10 curvas de ajuste C0 até C9, com valores de 100 até 1.000, com passos de tamanho 100. Foi obtido um pico de desempenho de cerca de 650 MCPS, que representa uma melhoria de 542 vezes no desempenho, quando comparado ao modelo ISS, que possui 1,20 MCPS.

A análise de desempenho em MIPS, considerando a figura 81, foram realizadas simulações com uma configuração de 1.000.000 iterações durante a

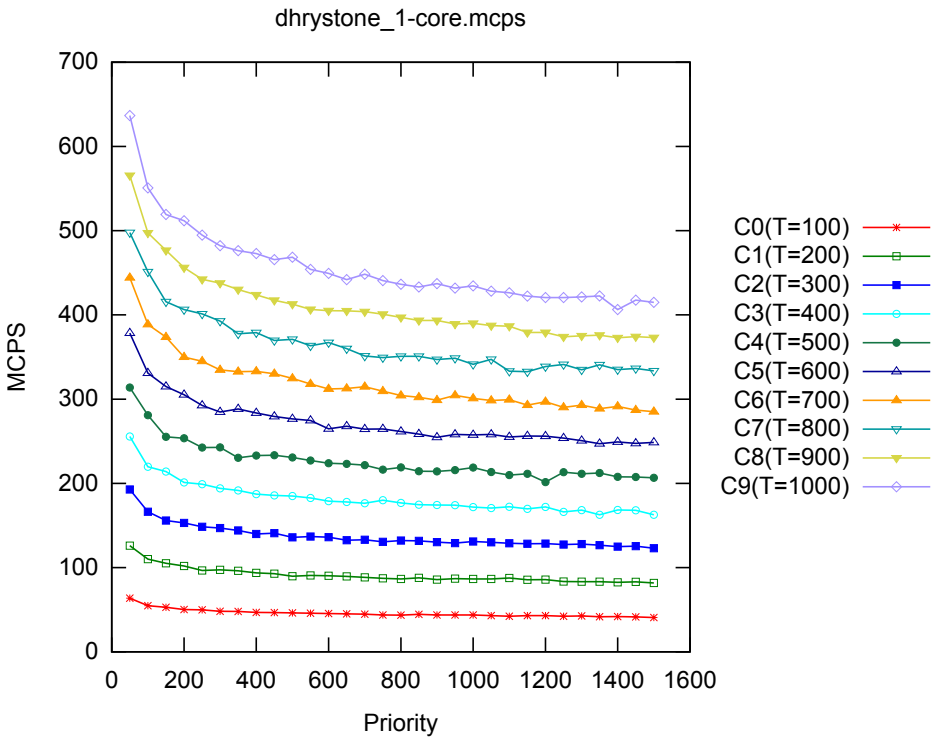


Figura 80: Desempenho em MCPS de Dhrystone

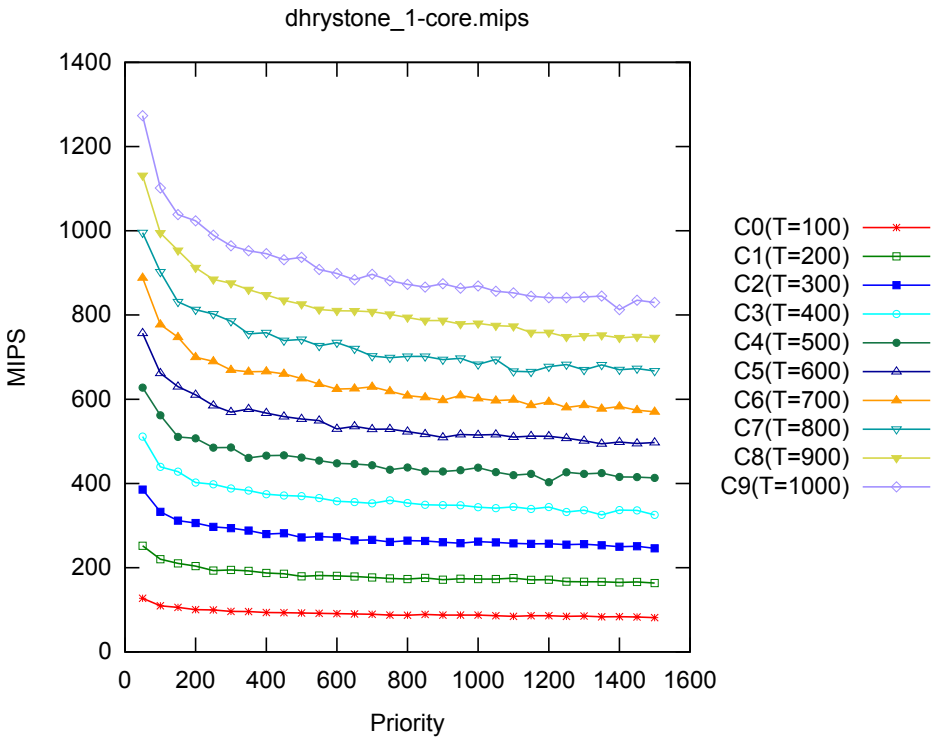


Figura 81: Desempenho em MIPS de Dhrystone

avaliação do desempenho. No modelo proposto foi atingido um pico de desempenho de cerca de 1.300 MIPS, no intervalo considerado, enquanto que no ISS foi obtido um desempenho de 0,68 MIPS. Estes resultados implicam em

uma melhoria de desempenho de aproximadamente 1.912 vezes no uso do modelo proposto.

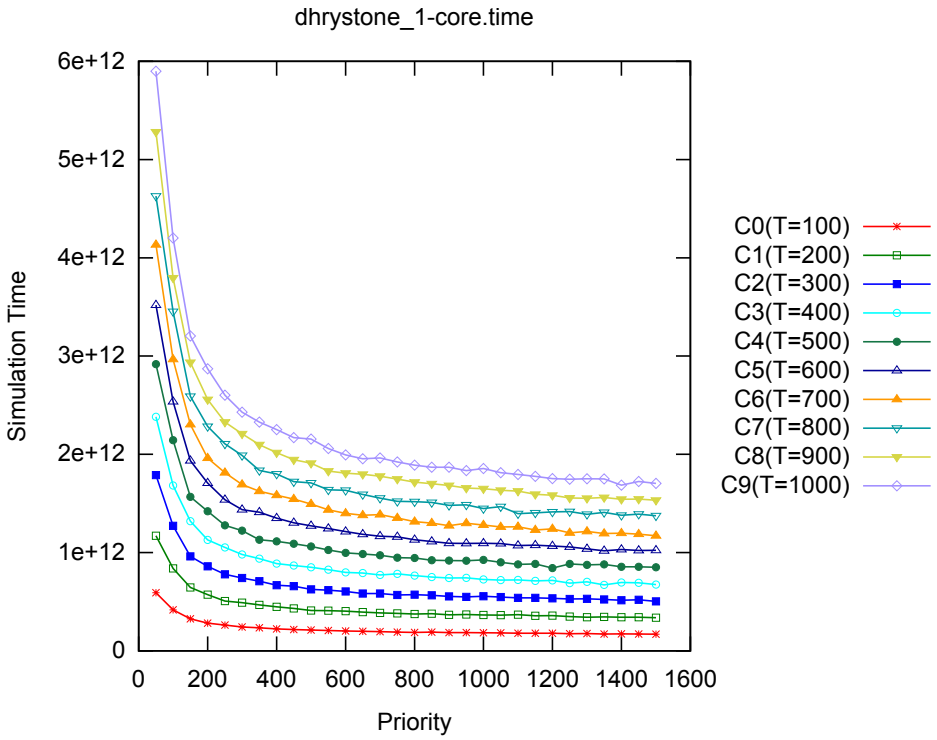


Figura 82: Tempo simulado de Dhrystone

Conforme previsto no modelo teórico de estimativa de tempo simulado, o comportamento temporal obtido pela aplicação foi bem previsível e segue a parametrização de prioridade e de ajuste com as curvas C0 até C9. A figura 82 deixa bem evidente este comportamento previsto, através da organização das curvas e seu comportamento teórico. Durante a sua simulação no modelo ISS, o Dhrystone teve um tempo simulado de 2,218809006e+12 picosegundos e mais uma vez é desejada uma parametrização que maximize o desempenho e reduza o erro relativo ao ISS. Através da análise do comportamento, foram calibrados parâmetros de ajuste com valor de 1.286 e de prioridade com valor de 1.084, obtendo um tempo simulado de 2,212767536401e+12 picosegundos com 3 simulações, com erro relativo ao ISS de 0,27%, desvio padrão de 1,8009920145e+10 picosegundos (±0,81%) e desempenho de 528,73 MCPS e 1.058 MIPS. A melhoria de desempenho obtida pela abordagem proposta com a parametrização acima é de aproximadamente 439 e 1.568 vezes, quando comparados aos resultados gerados pela simulação com ISS que foram 1,20 MCPS e 0,68 MIPS.

**Conclusões** Considerando a aplicação Dhrystone e a plataforma virtual de referência baseada em ISS, foi obtido um índice de desempenho de 0,257 DMIPS/MHz na execução da aplicação de referência em ambiente de simulação. A plataforma Aeroflex Gaisler LEON3 (70), que é baseada na arquitetura SPARC (64), alcançou o desempenho de 1,40 DMIPS/MHz (46) utilizando diferentes tecnologias de fabricação em hardware, como FPGA e ASIC. Este resultado demonstra que a plataforma de referência baseada em ISS possui desempenho cerca de 5,39 vezes mais lento, quando comparado com a implementação em hardware.

Para demonstrar a consistência das estimativas de desempenho, o parâmetro de ajuste é reduzido para 230, mantendo o parâmetro de prioridade, para configurar o modelo proposto para realizar a mesma quantidade de operações em tempo simulado cerca de 5 vezes menor, sendo obtido um desempenho de 1,335 DMIPS/MHz que é muito próximo da plataforma de hardware considerada. Diante dos resultados obtidos e das informações coletadas, é possível concluir que a abordagem proposta é capaz de fornecer estimativas de execução com baixo nível de erro e que são consistentes com os resultados observados em implementações em hardware consideradas, além de proporcionar ao projetista do sistema um ambiente de simulação com alto desempenho de simulação.

#### 6.4.2.3 Análise Comparativa

Esta subseção é dedicada a resumir e comentar com uma visão geral os resultados obtidos pelas aplicações de computação intensiva, focando nos aspectos de desempenho do modelo proposto HdSC e do modelo clássico ISS, na melhoria de desempenho obtida e no erro percentual das estimativas geradas. Em todas estas métricas, o ISS é considerado o ponto de referência para o cálculo da melhoria de desempenho e do erro relativo do tempo simulado obtido.

Nas tabelas a seguir, a primeira coluna diz respeito ao nome da aplicação em questão com seu tamanho de entrada entre parênteses, quando for aplicável. Nas demais colunas, todas as métricas consideradas são listadas em termos do desempenho do modelo proposto (HdSC) e do ISS, nas unidades de MCPS e MIPS, da melhoria de desempenho obtida pelo modelo proposto

sobre o ISS e do erro relativo das estimativas de tempo do modelo proposto com relação ao ISS.

Tabela 15: Comparativo de aplicações de computação intensiva

| Aplicação | HdSC | ISS | Desempenho | Erro |
|---|---|---|---|---|
| CoreMark | 362,07 MCPS<br>724,13 MIPS | 1,07 MCPS<br>0,65 MIPS | 340x<br>1.113x | 1,32% |
| Dhrystone | 528,73 MCPS<br>1.058 MIPS | 1,20 MCPS<br>0,68 MIPS | 439x<br>1.568x | 0,27% |

Na tabela 15, calculando-se o desempenho médio foi obtido aproximadamente 445,40 MCPS e 891,07 MIPS pelo modelo proposto (HdSC), enquanto que o ISS obteve uma média de 1,14 MCPS e 0,67 MIPS, com melhoria média de cerca de 391 e 1.330 vezes, respectivamente, com erro médio gerado de 0,80%.

Tabela 16: Comparativo de aplicações de computação intensiva (Automotive)

| Aplicação | HdSC | ISS | Desempenho | Erro |
|---|---|---|---|---|
| Basicmath (Pequeno) | 48.410 MCPS<br>96.819 MIPS | 0,91 MCPS<br>0,62 MIPS | 53.045x<br>156.945x | 2,66% |
| Basicmath (Grande) | 18.793 MCPS<br>37.585 MIPS | 0,95 MCPS<br>0,69 MIPS | 19.876x<br>59.659x | 15,03% |
| Bitcount (Pequeno) | 270,20 MCPS<br>540,40 MIPS | 1,32 MCPS<br>0,69 MIPS | 205x<br>780x | 4,99% |
| Bitcount (Grande) | 262,20 MCPS<br>524,40 MIPS | 1,33 MCPS<br>0,70 MIPS | 198x<br>750x | 1,61% |
| Quicksort (Pequeno) | 3.242 MCPS<br>6.484 MIPS | 0,94 MCPS<br>0,52 MIPS | 3.455x<br>12.468x | 17,29% |
| Quicksort (Grande) | 11.619 MCPS<br>23.238 MIPS | 0,98 MCPS<br>0,66 MIPS | 11.881x<br>35.231x | 14,39% |
| Susan (Pequeno) | 914,44 MCPS<br>1.829 MIPS | 1,04 MCPS<br>0,63 MIPS | 876x<br>2.893x | 0,52% |
| Susan (Grande) | 873,71 MCPS<br>1.747 MIPS | 1,07 MCPS<br>0,64 MIPS | 819x<br>2.727x | 1,59% |

Na tabela 16 foi calculado um desempenho médio da abordagem proposta (HdSC) de cerca de 10.548 MCPS e 21.096 MIPS, com o modelo ISS atingindo desempenho médio de 1,07 MCPS e 0,64 MIPS. A média de melhoria de desempenho foi de 9.858 e 32.963 vezes, respectivamente, com erro médio de 7,26%.

Na tabela 17, o desempenho médio obtido foi de 431,06 MCPS e 862,12

Tabela 17: Comparativo de aplicações de computação intensiva (Consumer)

| Aplicação | HdSC | ISS | Desempenho | Erro |
|---|---|---|---|---|
| JPEG Decoder (Pequeno) | 637,03 MCPS<br>1.274 MIPS | 1,21 MCPS<br>0,63 MIPS | 526x<br>2.034x | 0,78% |
| JPEG Decoder (Grande) | 632,99 MCPS<br>1.266 MIPS | 1,29 MCPS<br>0,66 MIPS | 491x<br>1.914x | 0,93% |
| JPEG Encoder (Pequeno) | 203,87 MCPS<br>407,75 MIPS | 1,31 MCPS<br>0,69 MIPS | 156x<br>593x | 0,99% |
| JPEG Encoder (Grande) | 215,36 MCPS<br>430,78 MIPS | 1,31 MCPS<br>0,69 MIPS | 165x<br>625x | 1,19% |
| Typeset (Pequeno) | 409,83 MCPS<br>819,66 MIPS | 1,27 MCPS<br>0,70 MIPS | 323x<br>1.176x | 1,22% |
| Typeset (Grande) | 487,27 MCPS<br>974,54 MIPS | 1,29 MCPS<br>0,71 MIPS | 378x<br>1.382x | 0,55% |

MIPS pelo modelo HdSC, enquanto que o modelo ISS obteve média de desempenho de 1,28 MCPS e 0,68 MIPS. Estes resultados representam uma melhoria média de 337 e 1.268 vezes, respectivamente, com erro médio de cerca de 0,94%.

Tabela 18: Comparativo de aplicações de computação intensiva (Network)

| Aplicação | HdSC | ISS | Desempenho | Erro |
|---|---|---|---|---|
| Dijkstra (Pequeno) | 193,80 MCPS<br>387,61 MIPS | 1,18 MCPS<br>0,73 MIPS | 163x<br>531x | 2,93% |
| Dijkstra (Grande) | 189,29 MCPS<br>378,58 MIPS | 1,21 MCPS<br>0,74 MIPS | 157x<br>511x | 1,91% |
| Patricia (Pequeno) | 4.922 MCPS<br>9.845 MIPS | 0,99 MCPS<br>0,65 MIPS | 4.953x<br>15.078x | 0,93% |
| Patricia (Grande) | 5.245 MCPS<br>10.489 MIPS | 0,98 MCPS<br>0,65 MIPS | 5.337x<br>16.230x | 0,81% |

Na tabela 18, o modelo proposto obteve uma média de desempenho de 2.638 MCPS e 5.275 MIPS que representam uma melhoria média de 2.420 e 7.645 vezes, respectivamente, quando comparadas as médias do ISS de 1,09 MCPS e 0,69 MIPS, com erro médio de 1,65%.

Na tabela 19 foi obtido um desempenho médio de 0,99 MCPS e 0,56 MIPS para o modelo ISS, enquanto que o modelo proposto obteve uma média de desempenho de 437,88 MCPS e 890,84 MIPS. Estes resultados contribuem em média para um aumento de 442 e 1.591 vezes, respectivamente, do desempenho obtido originalmente no ISS, com erro médio associado de 3,27%.

Na tabela 20, o modelo proposto atingiu um desempenho médio de 794,60

Tabela 19: Comparativo de aplicações de computação intensiva (Office)

| Aplicação | HdSC | ISS | Desempenho | Erro |
|---|---|---|---|---|
| Ispell (Pequeno) | 756,35 MCPS 1.513 MIPS | 1,23 MCPS 0,69 MIPS | 616x 2.188x | 1,26% |
| Ispell (Grande) | 575,97 MCPS 1.152 MIPS | 1,24 MCPS 0,70 MIPS | 466x 1.648x | 0,33% |
| Stringsearch (Pequeno) | 190,62 MCPS 381,24 MIPS | 0,36 MCPS 0,21 MIPS | 523x 1.862x | 9,06% |
| Stringsearch (Grande) | 228,56 MCPS 517,12 MIPS | 1,13 MCPS 0,64 MIPS | 229x 814x | 2,44% |

Tabela 20: Comparativo de aplicações de computação intensiva (Security)

| Aplicação | HdSC | ISS | Desempenho | Erro |
|---|---|---|---|---|
| Blowfish Decoder (Pequeno) | 565,31 MCPS 1.131 MIPS | 1,22 MCPS 0,64 MIPS | 464x 1.775x | 5,45% |
| Blowfish Decoder (Grande) | 554,71 MCPS 1.109 MIPS | 1,28 MCPS 0,67 MIPS | 435x 1.665x | 2,78% |
| Blowfish Encoder (Pequeno) | 547,47 MCPS 1.095 MIPS | 1,20 MCPS 0,63 MIPS | 456x 1.740x | 0,50% |
| Blowfish Encoder (Grande) | 554,43 MCPS 1.109 MIPS | 1,23 MCPS 0,64 MIPS | 451x 1.722x | 0,06% |
| Rijndael Decoder (Pequeno) | 1.210 MCPS 2.420 MIPS | 1,10 MCPS 0,65 MIPS | 1.097x 3.745x | 0,68% |
| Rijndael Decoder (Grande) | 1.273 MCPS 2.546 MIPS | 1,11 MCPS 0,65 MIPS | 1.143x 3.903x | 0,60% |
| Rijndael Encoder (Pequeno) | 1.173 MCPS 2.346 MIPS | 1,11 MCPS 0,65 MIPS | 1.055x 3.587x | 1,02% |
| Rijndael Encoder (Grande) | 1.252 MCPS 2.503 MIPS | 1,11 MCPS 0,65 MIPS | 1.129x 3.841x | 0,34% |
| SHA (Pequeno) | 397,61 MCPS 795,23 MIPS | 1,23 MCPS 0,65 MIPS | 323x 1.234x | 1,39% |
| SHA (Grande) | 418,51 MCPS 837,02 MIPS | 1,26 MCPS 0,66 MIPS | 333x 1.269x | 0,68% |

MCPS e 1.488 MIPS, com média de melhoria de cerca de 668 e 2.289 vezes, respectivamente, com relação ao modelo ISS que obteve média de 1,19 MCPS e 0,65 MIPS, com erro médio de 1,35%.

Tabela 21: Comparativo de aplicações de computação intensiva (Telecomm)

| Aplicação | HdSC | ISS | Desempenho | Erro |
|---|---|---|---|---|
| ADPCM Decoder (Pequeno) | 116,67 MCPS<br>233,34 MIPS | 1,25 MCPS<br>0,66 MIPS | 94x<br>356x | 0,40% |
| ADPCM Decoder (Grande) | 119,84 MCPS<br>239,67 MIPS | 1,27 MCPS<br>0,67 MIPS | 95x<br>360x | 0,67% |
| ADPCM Encoder (Pequeno) | 140,77 MCPS<br>281,53 MIPS | 1,27 MCPS<br>0,64 MIPS | 111x<br>441x | 0,67% |
| ADPCM Encoder (Grande) | 143,48 MCPS<br>286,95 MIPS | 1,35 MCPS<br>0,68 MIPS | 106x<br>425x | 0,80% |
| CRC32 (Pequeno) | 423,70 MCPS<br>847,41 MIPS | 1,35 MCPS<br>0,65 MIPS | 315x<br>1.307x | 4,38% |
| CRC32 (Grande) | 417,77 MCPS<br>835,54 MIPS | 1,32 MCPS<br>0,64 MIPS | 317x<br>1.315x | 0,82% |
| FFT (Pequeno) | 42.520 MCPS<br>85.040 MIPS | 0,95 MCPS<br>0,63 MIPS | 44.608x<br>134.154x | 8,44% |
| FFT (Grande) | 80.119 MCPS<br>160.239 MIPS | 0,94 MCPS<br>0,63 MIPS | 84.854x<br>253.342x | 0,97% |
| IFFT (Pequeno) | 50.375 MCPS<br>100.749 MIPS | 0,93 MCPS<br>0,62 MIPS | 54.120x<br>162.263x | 2,26% |
| IFFT (Grande) | 79.822 MCPS<br>159.644 MIPS | 0,95 MCPS<br>0,63 MIPS | 84.486x<br>252.322x | 0,51% |
| GSM Decoder (Pequeno) | 1.129 MCPS<br>2.258 MIPS | 1,07 MCPS<br>0,64 MIPS | 1.058x<br>3.528x | 4,23% |
| GSM Decoder (Grande) | 1.107 MCPS<br>2.215 MIPS | 1,11 MCPS<br>0,66 MIPS | 1.003x<br>3.348x | 0,20% |
| GSM Encoder (Pequeno) | 1.997 MCPS<br>3.993 MIPS | 0,94 MCPS<br>0,61 MIPS | 2.126x<br>6.534x | 4,57% |
| GSM Encoder (Grande) | 2.035 MCPS<br>4.070 MIPS | 0,97 MCPS<br>0,63 MIPS | 2.095x<br>6.432x | 0,40% |

Na tabela 21 foi obtida uma média de desempenho de 18.605 MCPS e 37.210 MIPS pelo modelo proposto que representa uma melhoria de 16.612 e 58.141 vezes, respectivamente, quando comparada a média do modelo ISS que foi de 1,12 MCPS e 0,64 MIPS, com erro médio de 2,09%.

### 6.4.3 Multiprocessada

Nos experimentos realizados com 2, 4, 8 e 16 processadores foi utilizada a mesma plataforma de referência ilustrada na figura 61. A única diferença existente é a presença de processadores simétricos adicionais que irão permitir que as aplicações fossem executadas concorrentemente. Todas as aplicações utilizadas também são as mesmas dos experimentos anteriores e de ambas classificações, sem modificações, uma vez que suportam múltiplos processadores.

#### 6.4.3.1 CoreMark

**Descrição** A aplicação de referência CoreMark tem como principal objetivo explorar ao máximo a capacidade de processamento de todos os processadores da plataforma, utilizando-se de algoritmos reais de processamento de listas, manipulação de matrizes, máquina de estados e checagem de CRC, por exemplo. Neste cenário de multiprocessamento, cada processador executa uma instância isolada da aplicação e no final do processamento, cada núcleo verifica a integridade dos dados gerados.

**Plataforma com 2 núcleos** Nas figuras 83 e 84 são ilustradas as curvas de desempenho médio de cada núcleo, considerando a métrica de MCPS e MIPS. Para a geração destes gráficos foi utilizada a configuração de 10 curvas de ajuste, com valores entre 100 e 1.000, com intervalo de tamanho 100, e de prioridade entre 50 e 1.500, com passos de tamanho 50. Pode ser visto um pico de desempenho próximo de cerca de 700 MCPS e 1.200 MIPS que representam uma melhoria de desempenho de aproximadamente 1.321 e 3.750 vezes, respectivamente, sob a simulação baseada em ISS que obteve 0,53 MCPS e 0,32 MIPS de desempenho médio por núcleo.

Analisando a figura 85 podem ser vistas as curvas de estimativa de tempo simulado geradas, considerando a parametrização definida anteriormente. É notório que o comportamental experimental reflete com exatidão o modelo teórico previsto e com a adição de um núcleo de processamento, este comportamento foi aprimorado. Na simulação com plataforma baseada em ISS foi obtido um tempo simulado de 1,1638747307e+13 picosegundos, com de-

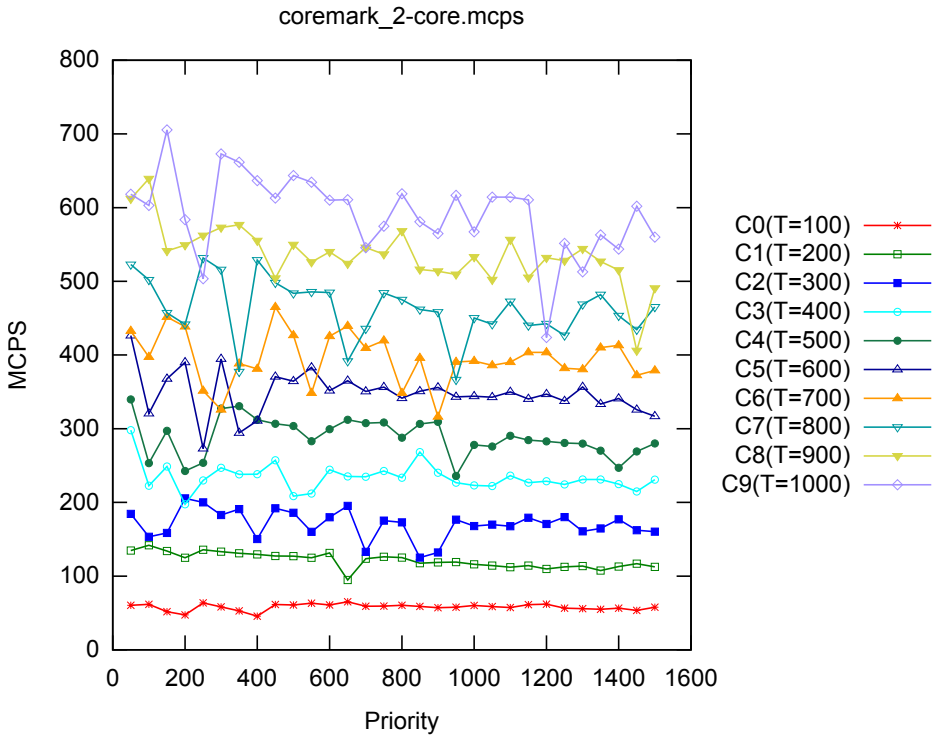


Figura 83: Desempenho em MCPS de CoreMark (2 núcleos)

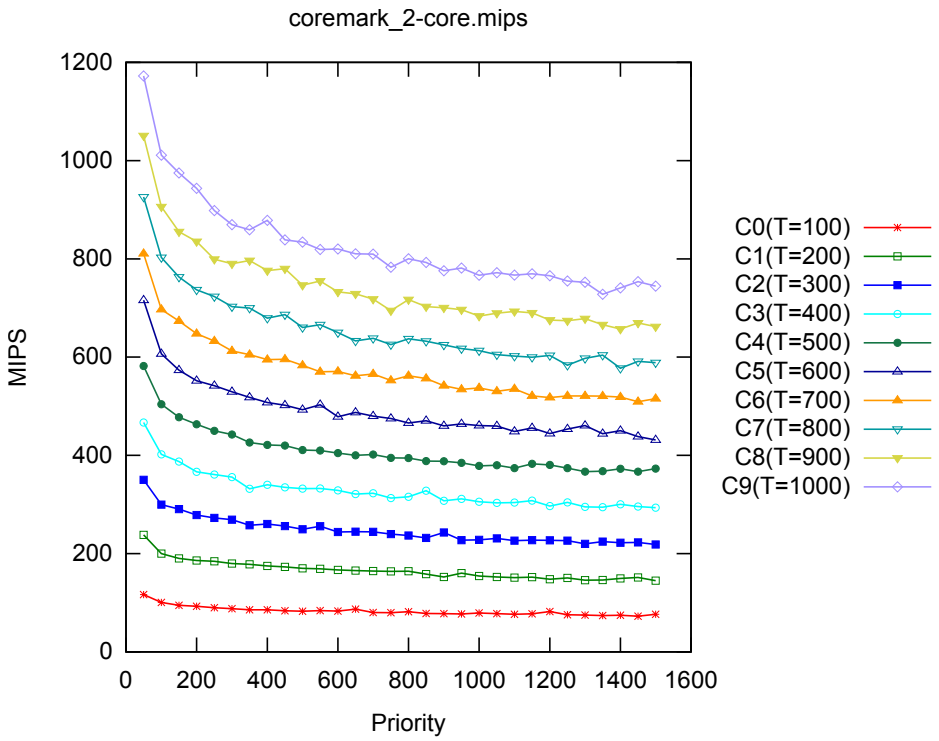


Figura 84: Desempenho em MIPS de CoreMark (2 núcleos)

sempenho médio por núcleo de 0,53 MCPS e 0,32 MIPS. Para que um comportamento equivalente seja observado na plataforma baseada no modelo proposto, são calibrados parâmetros de ajuste e de prioridade com valores

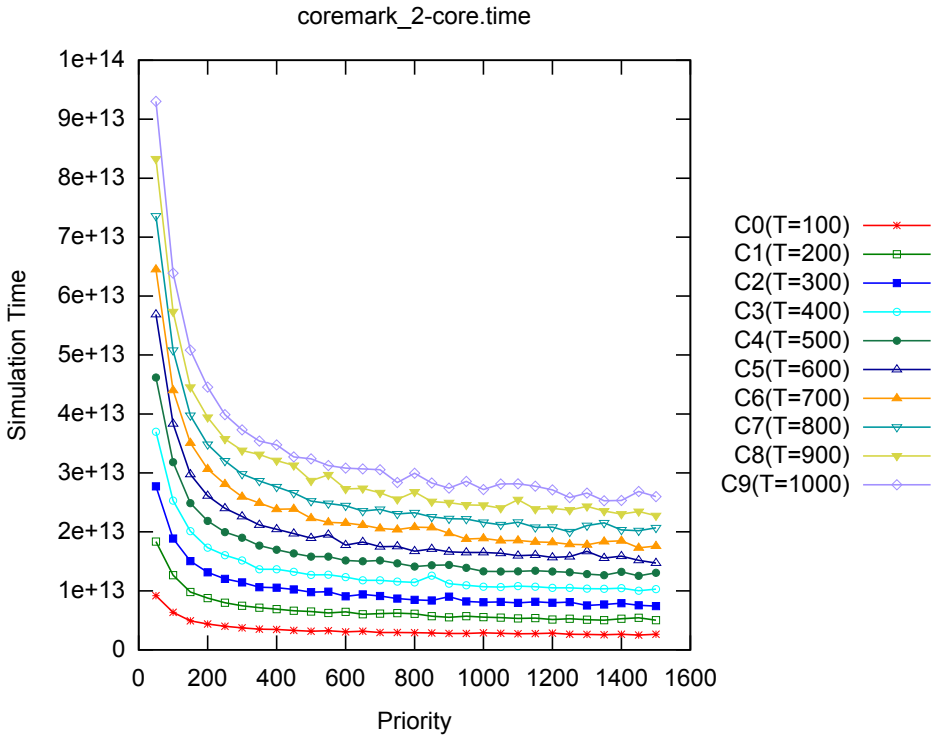


Figura 85: Tempo simulado de CoreMark (2 núcleos)

de 399 e 873, respectivamente, que geram uma estimativa de tempo simulado de 1,1728576576666e+13 picosegundos em 19 simulações, com erro relativo de 0,77% e desvio padrão de 1,6821676608e+11 picosegundos (±1,43%). O modelo proposto atinge 257,10 MCPS e 344,30 MIPS de desempenho, aumentando a velocidade de simulação em cerca de 489 e 1.073 vezes, quando comparado ao modelo ISS.

**Plataforma com 4 núcleos** Utilizando a mesma parametrização de ajuste e de prioridade definida anteriormente, é feita a geração dos gráficos de desempenho vistos nas figuras 86 e 87. Estes resultados refletem o desempenho médio de cada núcleo de processamento nas métricas MCPS e MIPS, atingindo picos de desempenho de aproximadamente 1.000 MCPS e 1.200 MIPS que representam um ganho de cerca de 4.000 e 8.000 vezes sob o modelo ISS que obteve uma média de 0,25 MCPS e 0,15 MIPS por núcleo.

Na figura 88 são ilustradas as curvas de estimativa de tempo simulado geradas para o CoreMark executando em 4 núcleos de processamento. Pode ser visto que mais uma vez o comportamento teórico previsto é observado e que a precisão dos resultados experimentais está muito próxima do resultado teórico. Na análise do relatório de simulação com ISS foi apurado um tempo si-

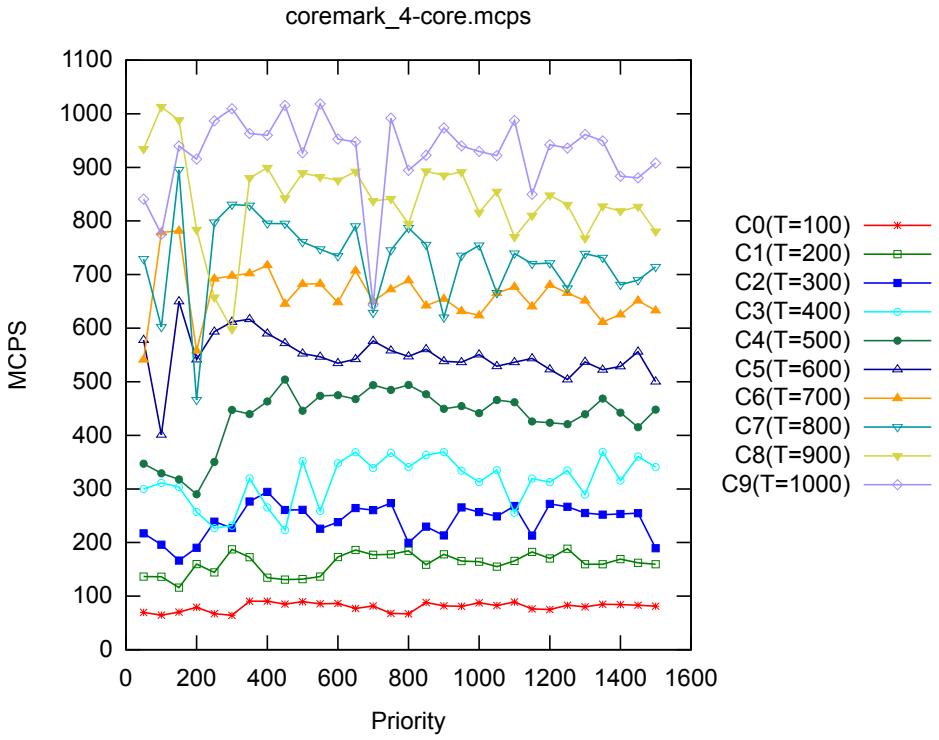


Figura 86: Desempenho em MCPS de CoreMark (4 núcleos)

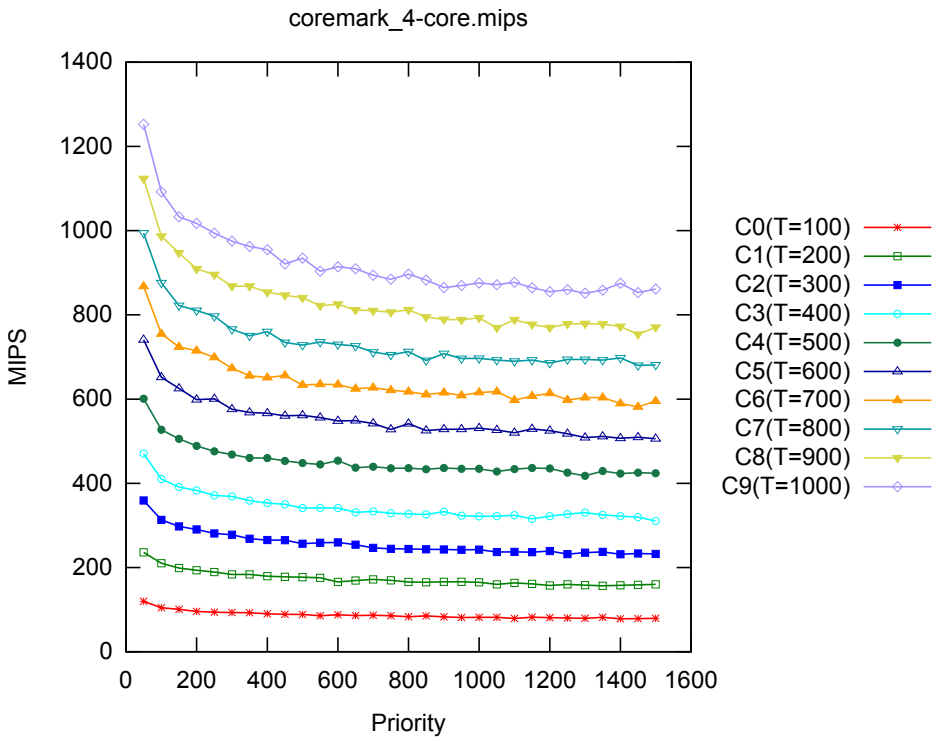


Figura 87: Desempenho em MIPS de CoreMark (4 núcleos)

mulado de 1,1638921744e+13 picosegundos, com desempenho médio de 0,25 MCPS e 0,15 MIPS por núcleo. Para equiparar este comportamento na plataforma com o modelo proposto, foram calibrados parâmetros de ajuste e de

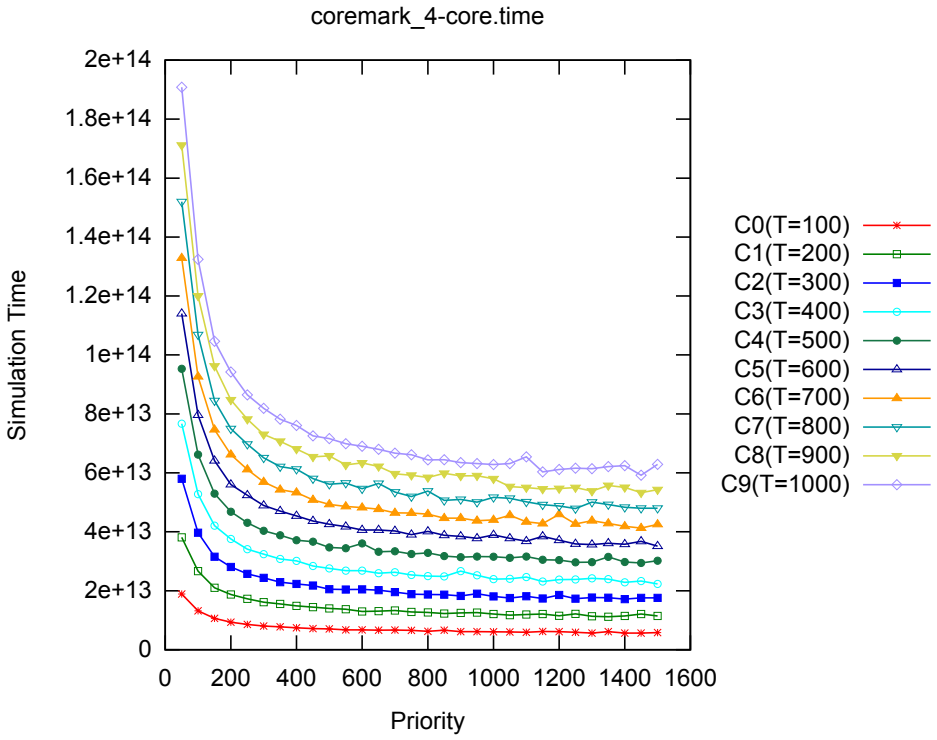


Figura 88: Tempo simulado de CoreMark (4 núcleos)

prioridade com valores de 185 e 937, respectivamente, que fornecem uma estimativa de tempo simulado de 1,1576408394676e+13 picosegundos com 11 simulações. Esta simulação atinge 165,99 MCPS e 157,50 MIPS de desempenho médio por núcleo, aumentando em cerca de 665 e 1.034 vezes a velocidade quando comparada ao ISS, com erro relativo de 0,53% e desvio padrão de 6,0835722094e+10 picosegundos (±0,52%).

**Plataforma com 8 núcleos** Com a mesma parametrização de ajuste e prioridade já descritas anteriormente, é feita a geração dos gráficos de desempenho médio em MCPS e MIPS para cada núcleo de processamento da plataforma, como pode ser visto nas figuras 89 e 90. São observados picos de desempenho de cerca de 1.000 MCPS e 1.050 MIPS que melhoram a velocidade de execução do sistema em aproximadamente 7.692 e 13.125 vezes, respectivamente, em comparação ao ISS que obteve 0,13 MCPS e 0,08 MIPS.

As estimativas de tempo simulado geradas com 8 núcleos de processamento podem ser visualizadas na figura 91. As curvas revelam o padrão previsto pelo modelo matemático, com bastante uniformidade e precisão. Nos resultados de simulação com a plataforma baseada em ISS foi obtido um tempo simulado de 1,1638931485e+13 picosegundos, com desempenho mé-

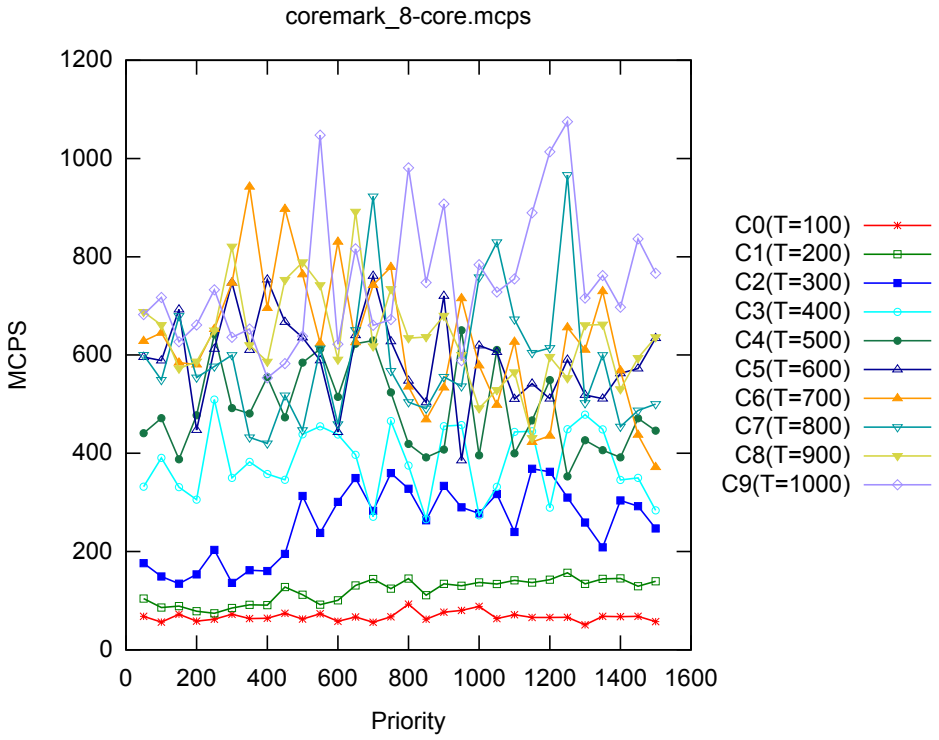


Figura 89: Desempenho em MCPS de CoreMark (8 núcleos)

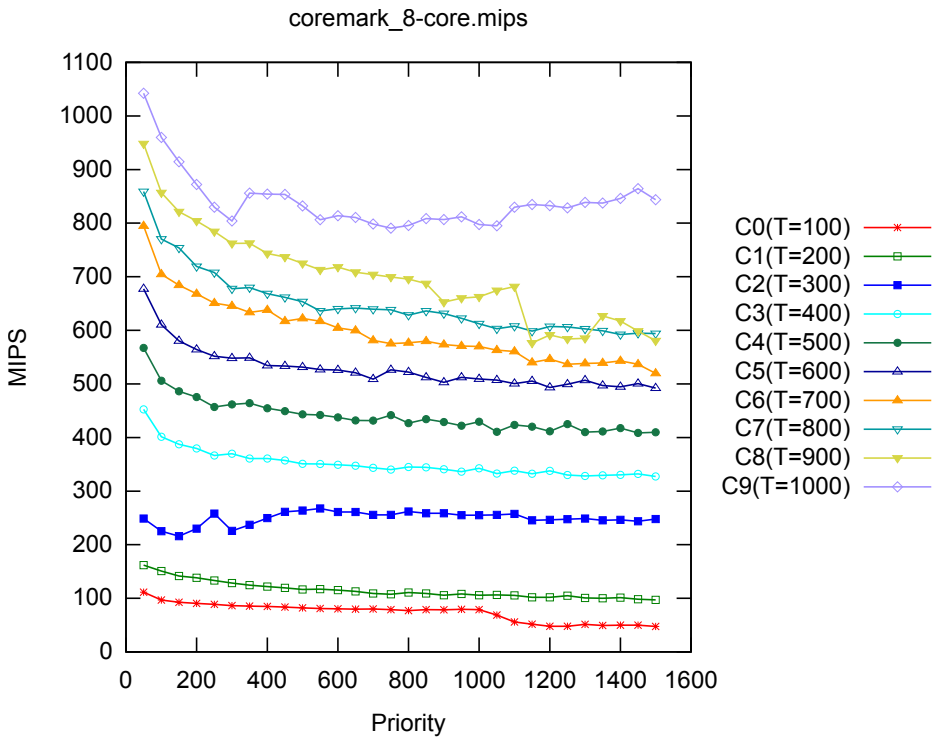


Figura 90: Desempenho em MIPS de CoreMark (8 núcleos)

dio de 0,13 MCPS e 0,08 MIPS por núcleo. Na configuração do modelo proposto para obter comportamento análogo, foram utilizados parâmetros de ajuste e de prioridade com valores de 80 e 970 que geram uma estimativa

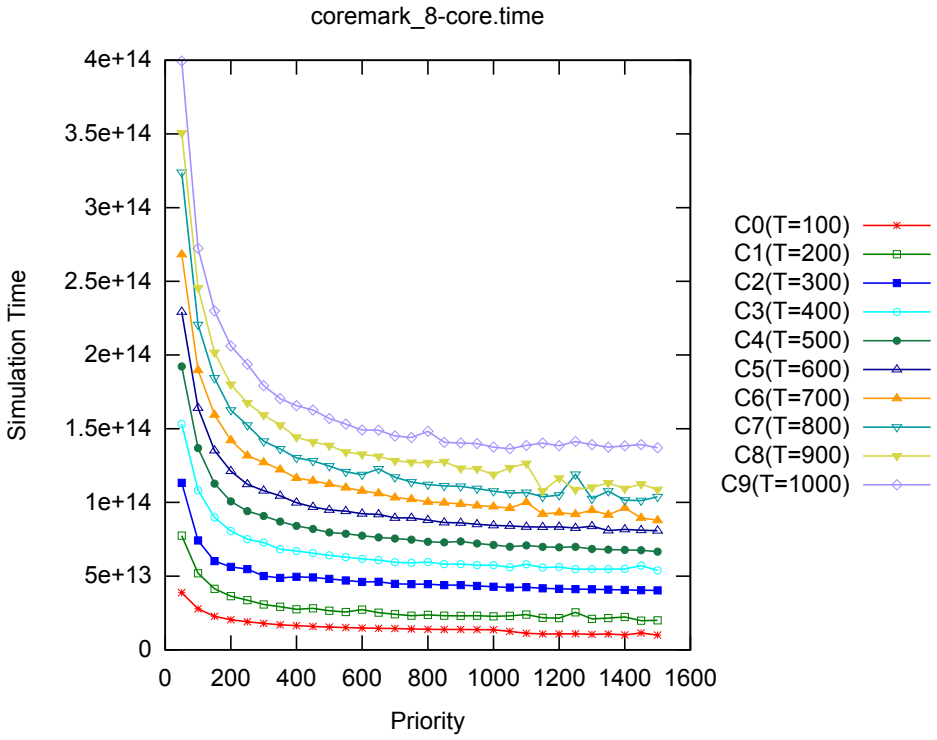


Figura 91: Tempo simulado de CoreMark (8 núcleos)

de tempo simulado de 1,1712456400684e+13 picosegundos com 9 simulações. Este resultado possui um erro relativo ao ISS de cerca de 0,63% e desvio padrão de 5,0856446734e+10 picosegundos (±0,43%), com desempenho médio de 96,11 MCPS e 72,45 MIPS que representam uma melhoria de aproximadamente 756 e 934 vezes, em comparação a plataforma baseada em ISS.

**Plataforma com 16 núcleos** Adotando as mesmas configurações de parâmetros já descrita anteriormente, são geradas as curvas de desempenho médio nas métricas de MCPS e MIPS para a aplicação CoreMark executando em 16 núcleos de processamento. Foi observado um pico de desempenho de cerca de 1.600 MCPS e 1.100 MIPS que representam um aumento de velocidade de execução de aproximadamente 26.667 e 27.500 vezes com relação do desempenho do ISS que foi de 0,06 MCPS e 0,04 MIPS.

Na figura 94 as curvas com as estimativas de tempo simulado geradas pode ver visualizadas, confirmando mais uma vez o comportamento teórico previsto e a alta precisão das estimativas geradas. Vale ressaltar que como esta simulação é extremamente longa, diferentes situações de carga são criadas, gerando variações na constante da máquina que está realizando as simulações. Realizando a simulação do CoreMark em uma plataforma com

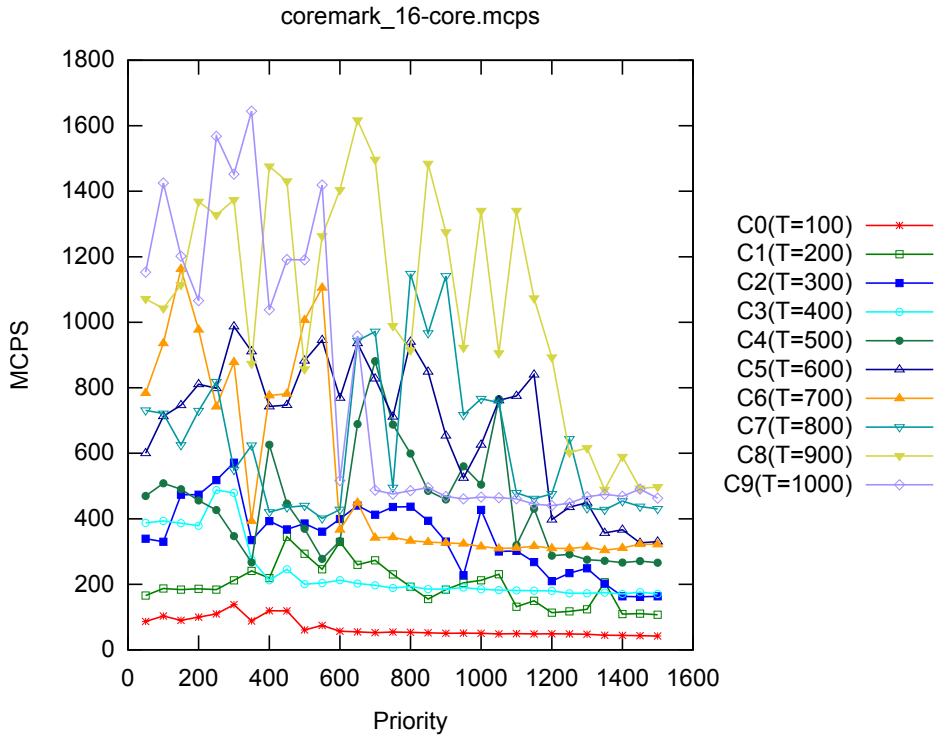


Figura 92: Desempenho em MCPS de CoreMark (16 núcleos)

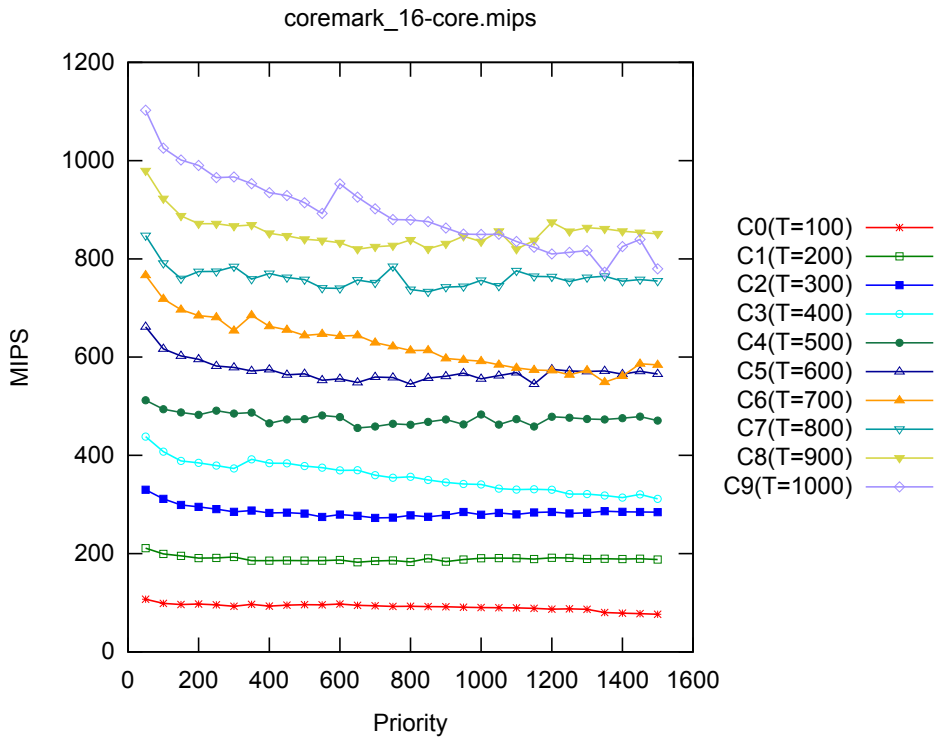


Figura 93: Desempenho em MIPS de CoreMark (16 núcleos)

núcleos ISS foi obtido um tempo simulado de 1,1639140354e+13 picosegundos, com desempenho médio de 0,06 MCPS e 0,04 MIPS por núcleo de processamento. Para atingir um comportamento equivalente, uma plataforma

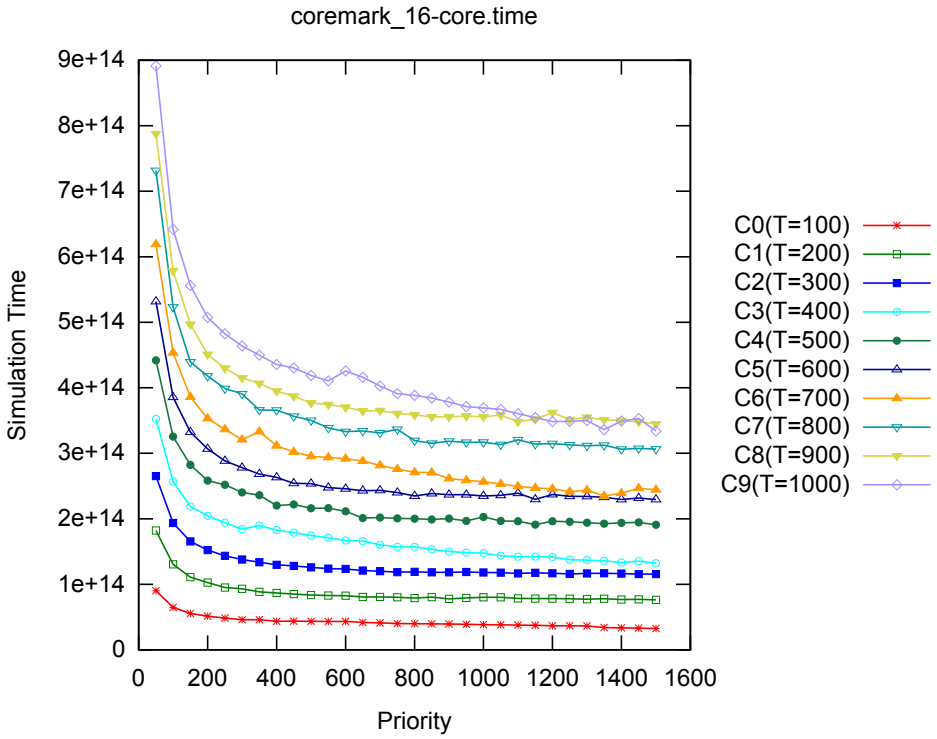


Figura 94: Tempo simulado de CoreMark (16 núcleos)

composta pelo modelo proposto foi configurada com parâmetros de ajuste e de prioridade com valores de 31 e 902 que estimam um tempo simulado de 1,1770974078853e+13 picosegundos em 19 simulações, com erro relativo de 1,13% e desvio padrão de 6,1634754319e+10 picosegundos (±0,52%). O desempenho médio obtido foi de 37,87 MCPS e 28,72 MIPS por núcleo, o que representa um aumento de desempenho de 608 e 756 vezes em relação ao modelo ISS.

**Conclusões** No estudo de caso realizado com o CoreMark pode ser visto a aplicação com alto nível de utilização de processamento e algoritmos reais utilizados pela indústria. O principal objetivo foi ilustrar a capacidade da abordagem proposta de executar sistemas com múltiplos processadores que façam o uso intensivo dos recursos computacionais e ainda sim o comportamento teórico previsto é observado. Além da aderência ao modelo de tempo concebido, o núcleo de simulação implementado de forma paralela (POSIX Threads) permite a extração do máximo de recursos do sistema de desenvolvimento, proporcionando índices elevados de desempenho.

#### 6.4.3.2 Device Control

**Descrição** O Device Control é uma aplicação desenvolvida para realizar gerenciamento de dispositivos de contagem de tempo da plataforma. O seu funcionamento está atrelado a eventos gerados pelos dispositivos e a execução só é finalizada quando um determinado tempo de simulação foi atingido. Independentemente dos parâmetros de configuração utilizados, a aplicação possui suas próprias restrições temporais e irá finalizar sua execução em um tempo simulado pré-determinado.

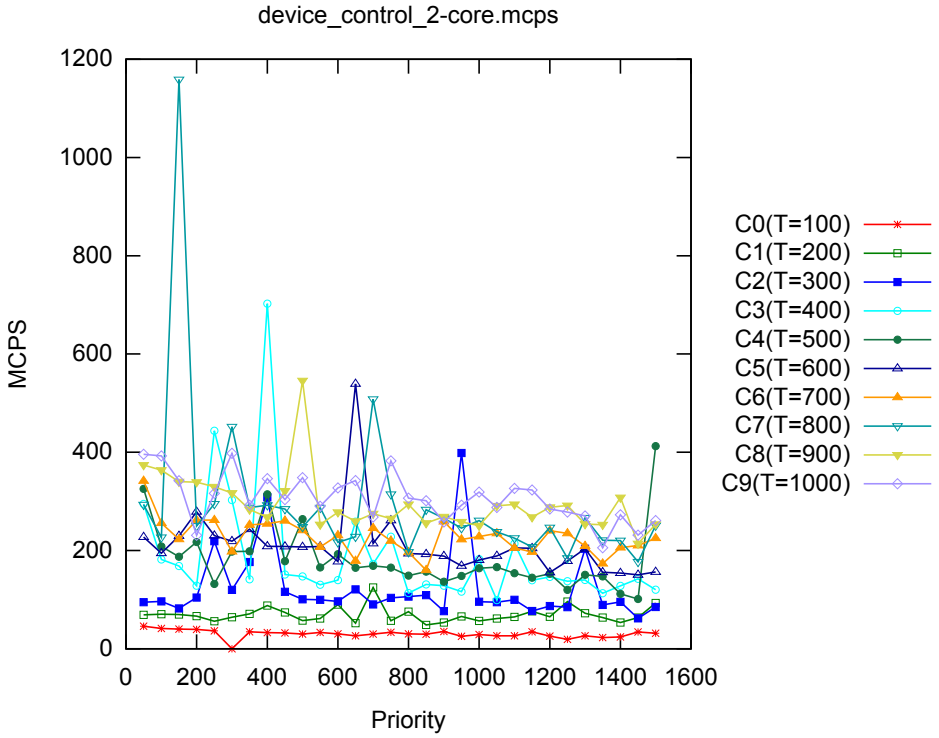


Figura 95: Desempenho em MCPS de Device Control (2 núcleos)

**Plataforma com 2 núcleos** Para realização dos experimentos foram definidos parâmetros de configuração de ajuste com 10 curvas (C0 até C9), com valores entre 100 e 1.000, com intervalos de tamanho 100, e de prioridade com valores entre 50 e 1.500, com passos de tamanho 50. Com esta parametrização foram gerados os gráficos de desempenho em MCPS e MIPS para a aplicação Device Control, como pode ser visto nas figuras 95 e 96. Com um pico de desempenho de aproximadamente 1.200 MCPS e 550 MIPS, o modelo proposto executa cerca de 2.727 e 3.667 vezes mais rápido, respectivamente, que o ISS que atingiu 0,44 MCPS e 0,15 MIPS.

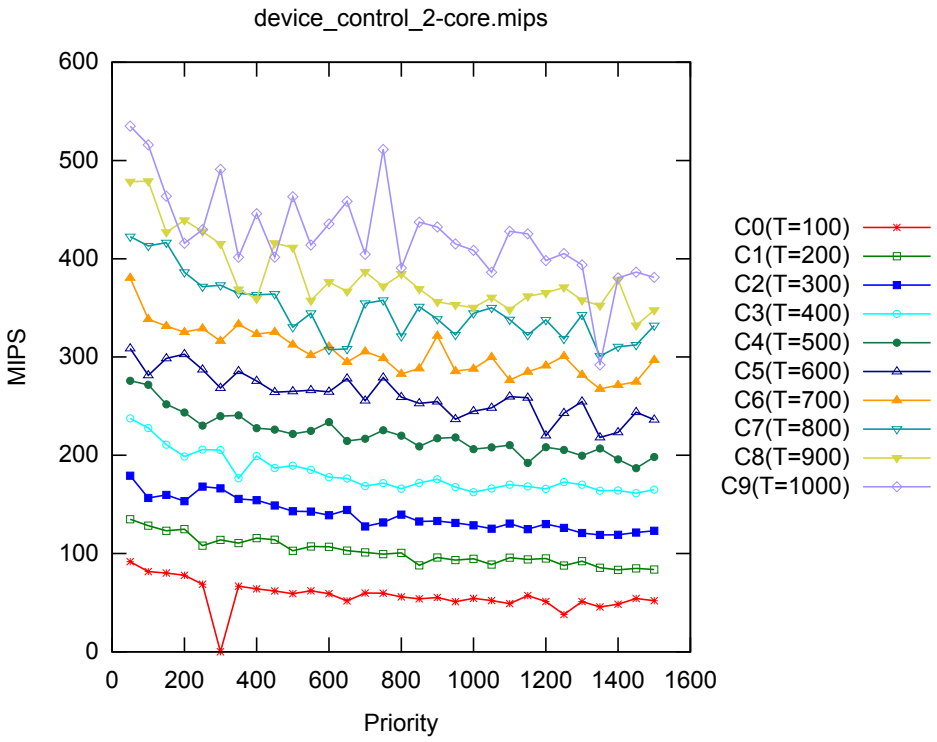


Figura 96: Desempenho em MIPS de Device Control (2 núcleos)

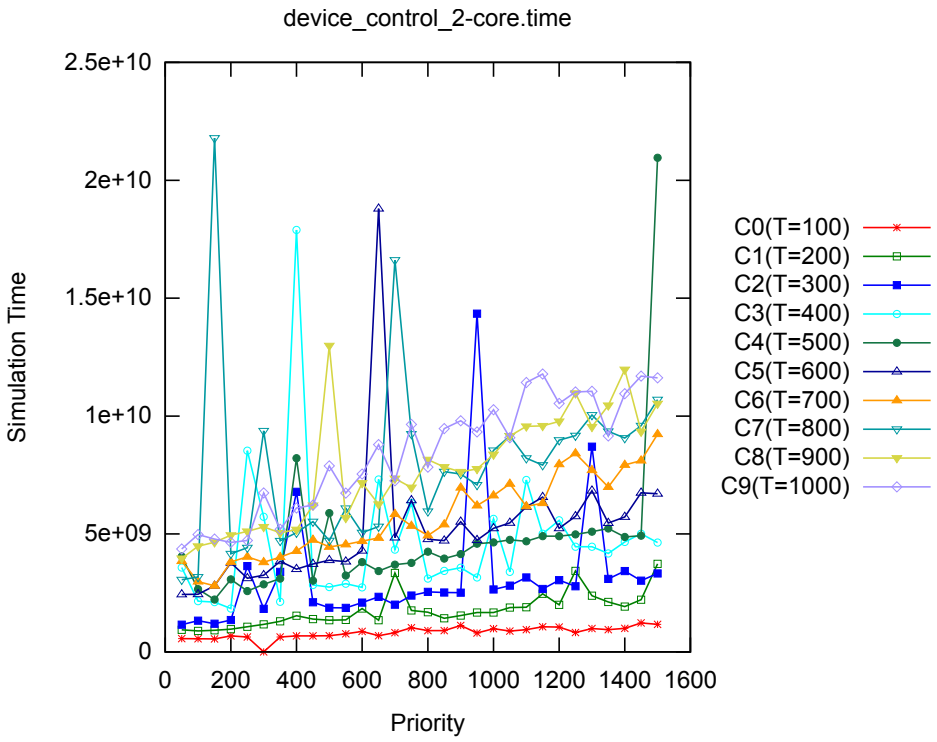


Figura 97: Tempo simulado de Device Control (2 núcleos)

Na figura 97 são exibidas as curvas de estimativa de tempo simulado geradas utilizando a parametrização definida anteriormente. Apesar do modelo teórico prever um comportamento de tempo simulado, a aplicação em si

possui uma restrição de tempo para finalizar sua execução. Assim o modelo teórico não é invalidado nem é criada uma exceção, tendo em vista que a aplicação tem como parte do seu comportamento um tempo simulado fixo e conhecido durante sua execução. Calibrando os parâmetros de ajuste e de prioridade com valores de 136 e 1.291, respectivamente, é obtida uma estimativa de tempo simulado de 1,657965877e+9 picosegundos em 3.519 simulações, com erro relativo ao ISS de 10,55% e desvio padrão de 5,91493481e+8 picosegundos (±35,67%). Com desempenho de 45,94 MCPS e 67,54 MIPS, a abordagem proposta é cerca de 115 e 360 vezes mais rápida, respectivamente, em comparação com o modelo ISS.

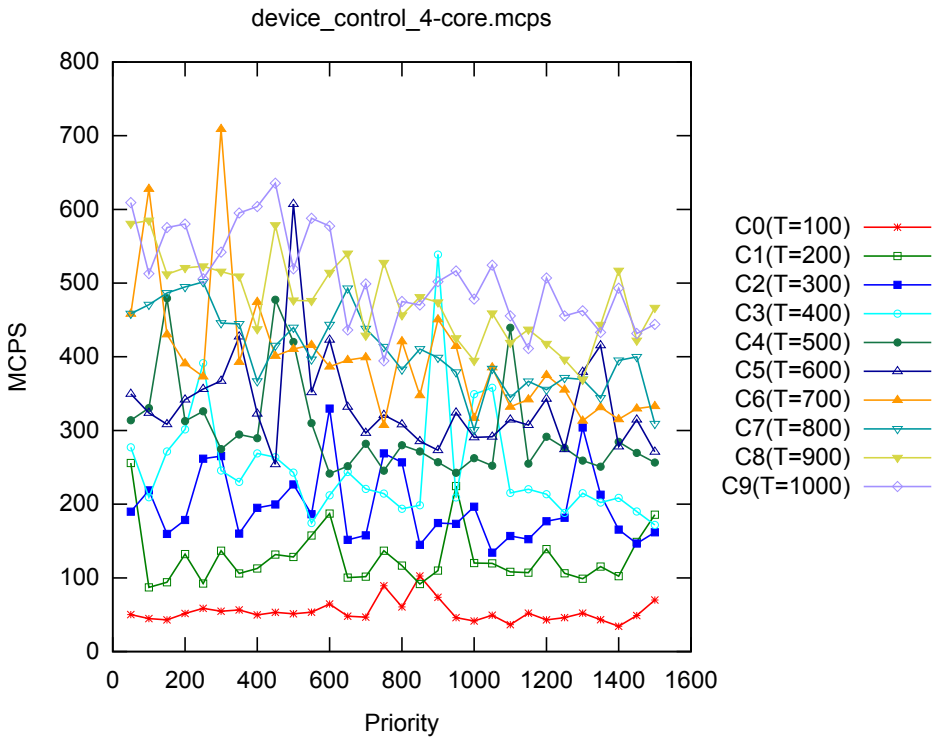


Figura 98: Desempenho em MCPS de Device Control (4 núcleos)

**Plataforma com 4 núcleos** Adotando a mesma parametrização de curvas de ajuste e de prioridade, são gerados os gráficos de desempenho em MCPS e MIPS para a aplicação Device Control executando em 4 núcleos de processamento, como pode ser visto nas figuras 98 e 99. Com um pico de desempenho médio de aproximadamente 700 MCPS e 700 MIPS por núcleo, a abordagem proposta é cerca de 3.684 e 5.833 vezes mais rápido que o modelo ISS que obteve desempenho médio de 0,19 MCPS e 0,12 MIPS por núcleo.

Na figura 100 são exibidas as estimativas de tempo simulado do modelo

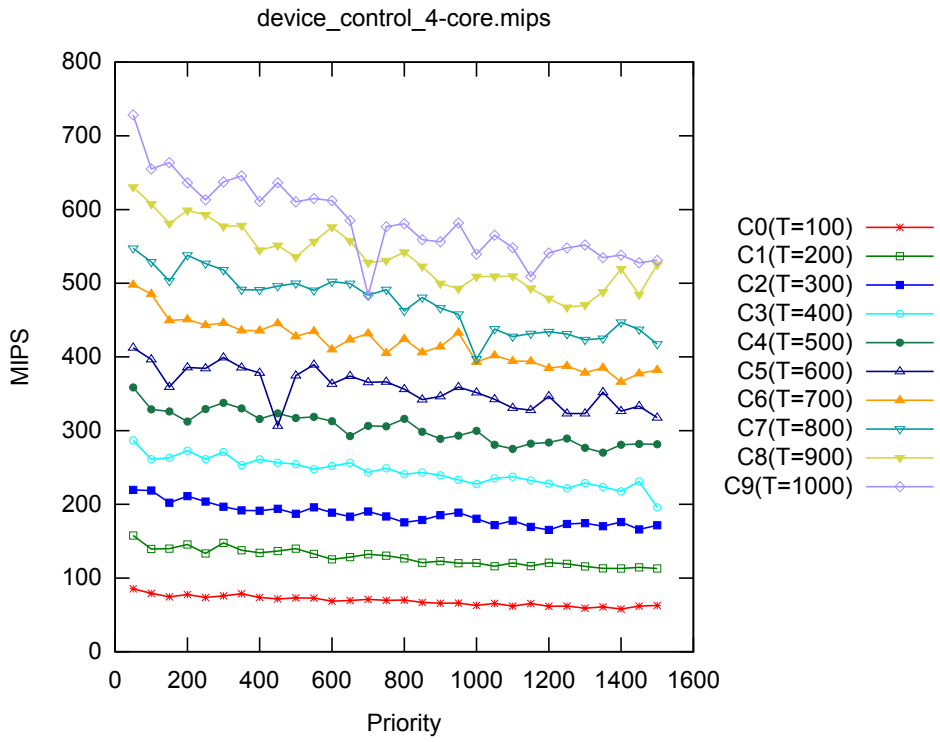


Figura 99: Desempenho em MIPS de Device Control (4 núcleos)

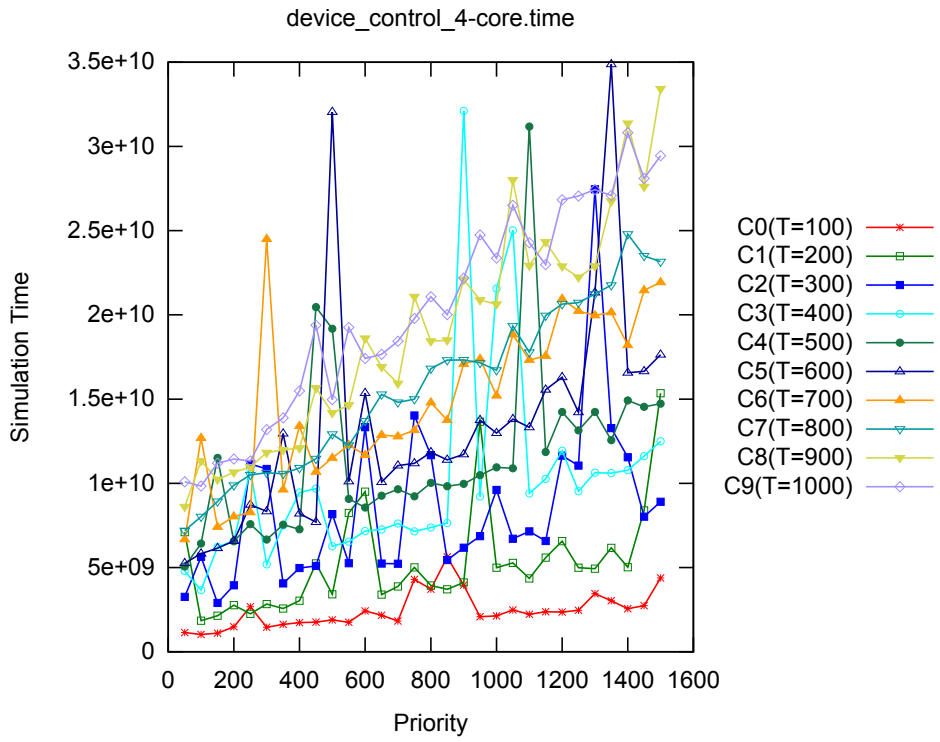


Figura 100: Tempo simulado de Device Control (4 núcleos)

proposto para a aplicação Device Control executando em uma plataforma com 4 núcleos de processamento. Realizando a simulação em uma plataforma plataforma baseada em ISS foi obtido um tempo simulado de 1,085279e+9

picosegundos, com desempenho médio de 0,19 MCPS e 0,12 MIPS por núcleo de processamento. Na configuração dos parâmetros de ajuste e de prioridade do modelo proposto foram calibrados os valores 50 e 753, respectivamente, que geram uma estimativa de tempo simulado de 1,174114982e+9 picosegundos em 538 simulações, com erro relativo ao ISS de 8,18% e desvio padrão de 1,43766905e+8 picosegundos (±12,24%). Atingindo um desempenho médio de 20,12 MCPS e 34,03 MIPS, o modelo proposto é cerca de 104 e 317 vezes mais rápido do que o modelo ISS.

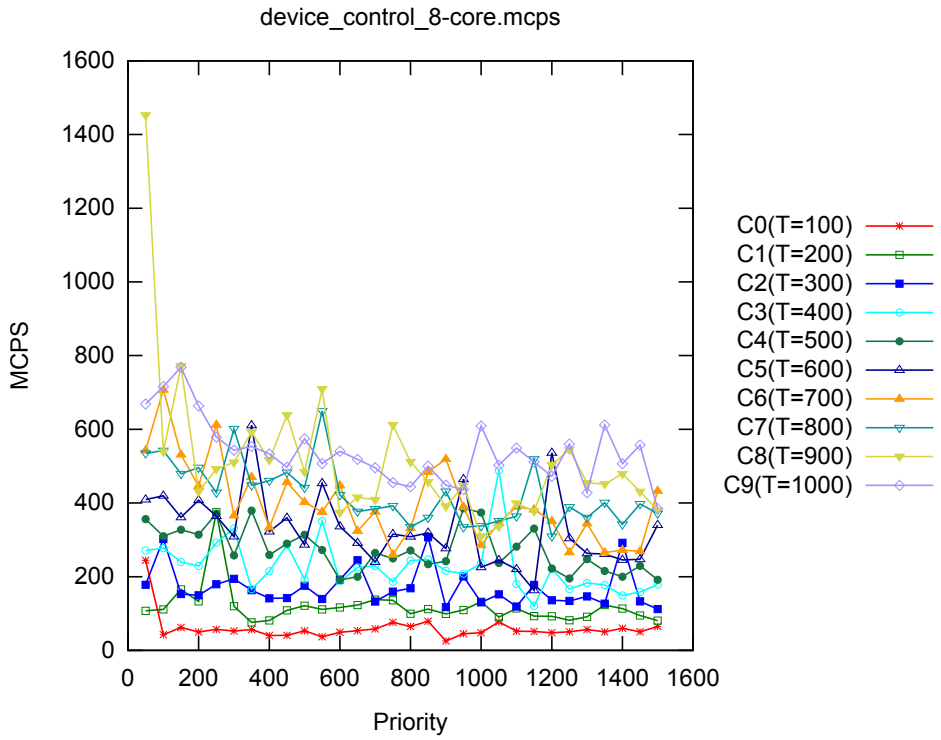


Figura 101: Desempenho em MCPS de Device Control (8 núcleos)

**Plataforma com 8 núcleos** Nas figuras 101 e 102 são exibidas as curvas de desempenho médio em MCPS e MIPS, respectivamente, para a aplicação Device Control executando em 8 núcleos de processamento, considerando a mesma parametrização já descrita anteriormente. Pode ser visto um pico de desempenho médio de cerca de 1.400 MCPS e 650 MIPS por núcleo que representa um aumento de velocidade de aproximadamente 11.667 e 10.833 vezes em comparação ao núcleo ISS que atingiu média de desempenho de 0,12 MCPS e 0,06 MIPS.

Com o aumento do número de núcleos de processamento, as estimativas de tempo geradas começam a exibir um comportamento mais ruidoso, como

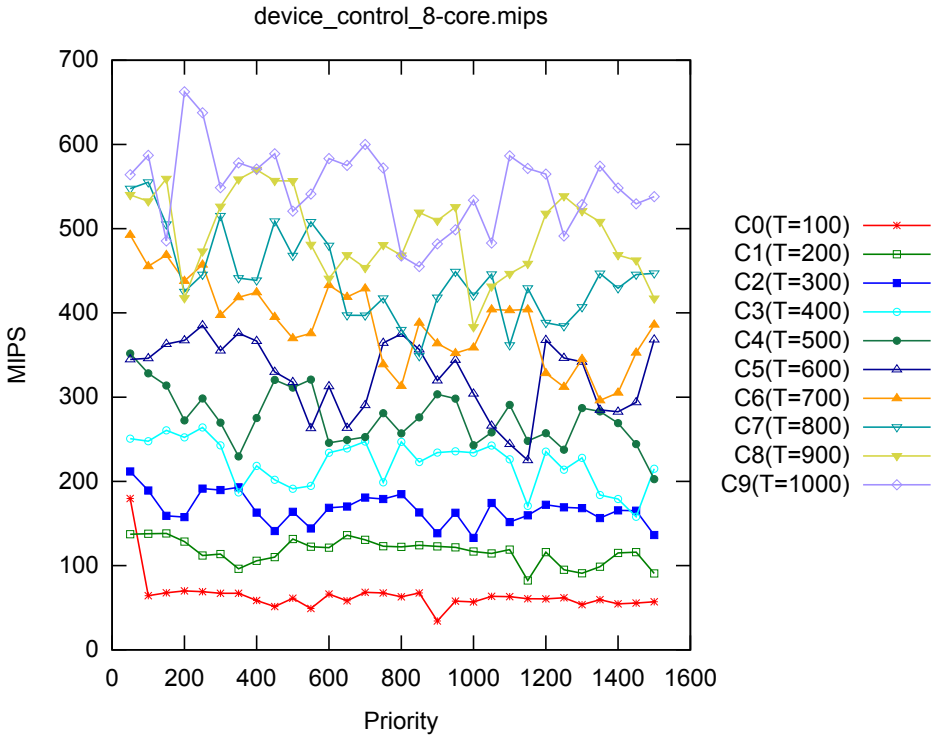


Figura 102: Desempenho em MIPS de Device Control (8 núcleos)

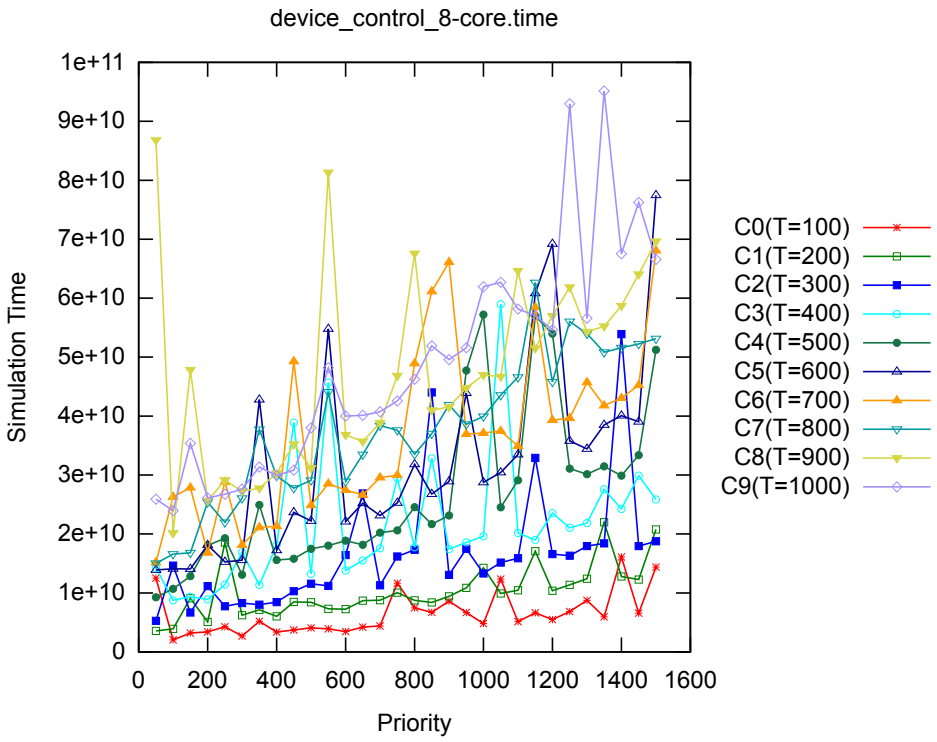


Figura 103: Tempo simulado de Device Control (8 núcleos)

pode ser visto na figura 103. Durante a simulação utilizando processadores baseados em ISS foi atingido um tempo simulado de 1,184342e+9 picosegundos, com desempenho médio por núcleo de 0,12 MCPS e 0,06 MIPS. Realizando a

simulação com o modelo proposto, foram calibrados parâmetros de ajuste e de prioridade com valores de 17 e 946 que geram uma estimativa de tempo simulado de 1,106861784e+9 picosegundos em 547 simulações, com desempenho médio de 7,08 MCPS e 11,85 MIPS por núcleo. Estes resultados apresentam um erro relativo de 6,54%, com um desvio padrão de 1,28445749e+8 picosegundos (±11,60%) e aumento de desempenho de cerca de 58 e 184 vezes, considerando métricas MCPS e MIPS, respectivamente, em comparação com o modelo ISS.

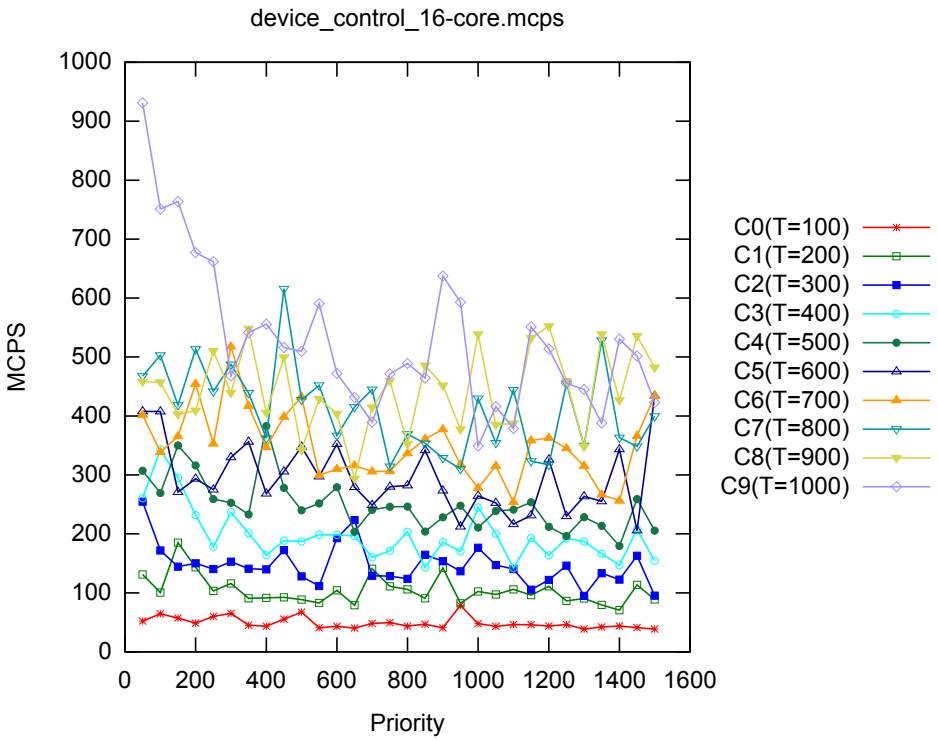


Figura 104: Desempenho em MCPS de Device Control (16 núcleos)

**Plataforma com 16 núcleos** Considerando a parametrização descrita e uma plataforma com 16 núcleos de processamento, são gerados os gráficos de desempenho em MCPS e MIPS para a aplicação Device Control, como pode ser visto nas figuras 104 e 105. São obtidos picos de desempenho de aproximadamente 900 MCPS e 700 MIPS que representam um aumento de velocidade de execução de cerca de 15.000 e 23.333 vezes, respectivamente, em comparação com o modelo ISS que obteve 0,06 MCPS e 0,03 MIPS.

Na análise das estimativas de tempo simulado geradas, vistas na figura 106, são observados basicamente o mesmo comportamento visto nas simulações anteriores. Isto se deve ao fato do número adicional de núcleos não

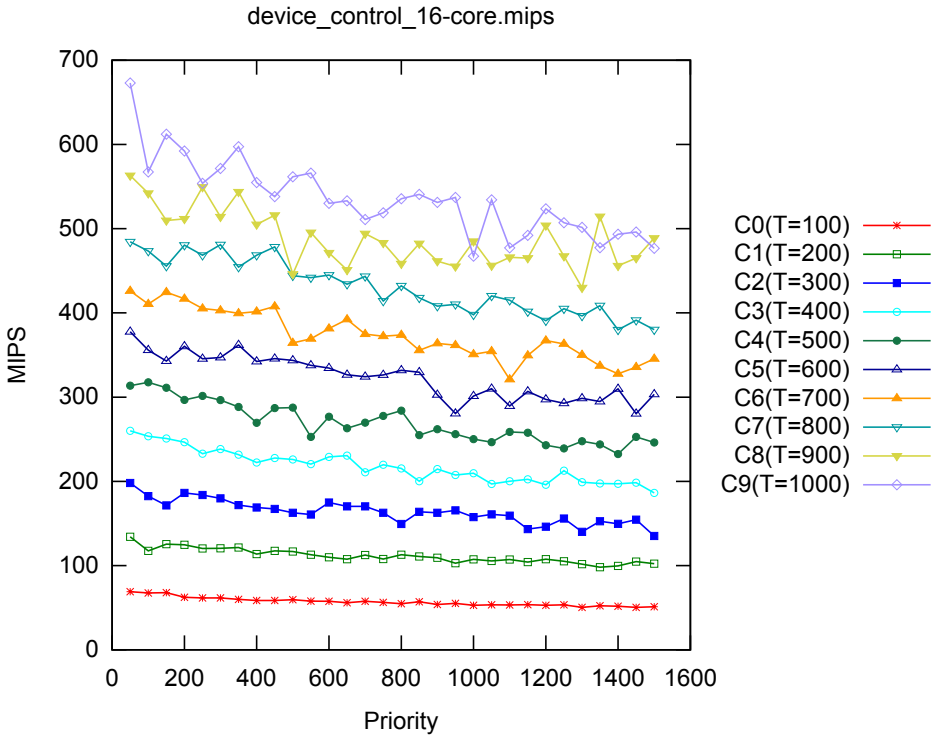


Figura 105: Desempenho em MIPS de Device Control (16 núcleos)

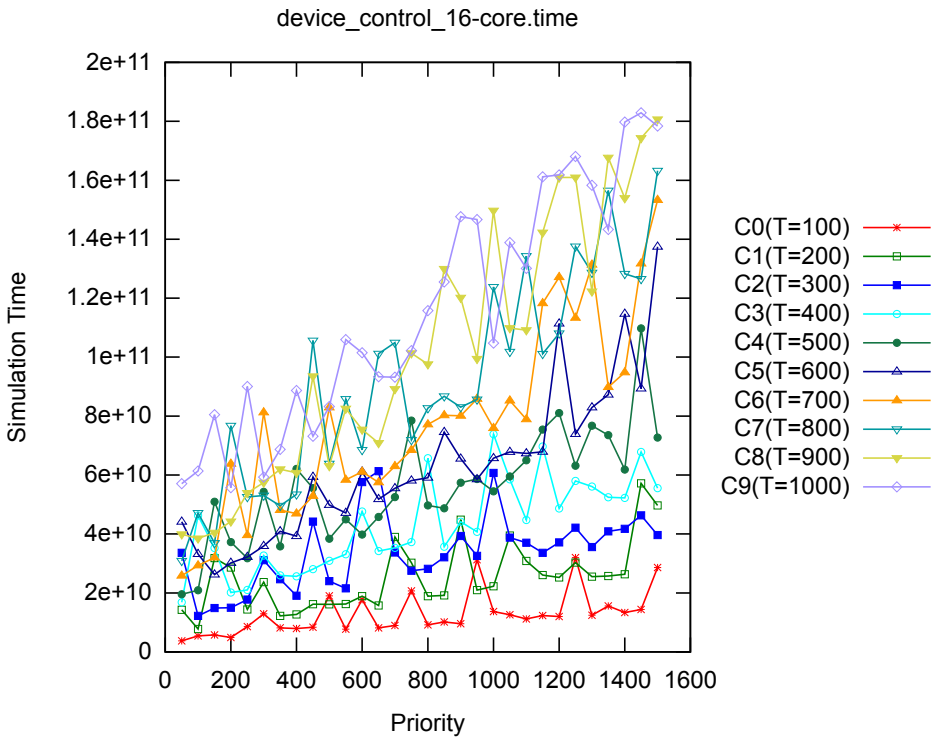


Figura 106: Tempo simulado de Device Control (16 núcleos)

interferir no tempo simulado da aplicação que é fixo. Realizando os experimentos com um plataforma baseada em ISS foi atingido um tempo simulado de 1,091539e+9 picosegundos, com desempenho médio de 0,06 MCPS e 0,03

MIPS por núcleo de processamento. Utilizando o modelo proposto na plataforma, foram definidos parâmetros de ajuste e de prioridade com valores de 7 e 993 que geram uma estimativa de tempo simulado de 1,135215694e+9 picosegundos em 683 simulações, com erro relativo ao ISS de 4,00% e desvio padrão de 1,37238214e+8 picosegundos (±12,08%). O desempenho médio obtido foi de 3,15 MCPS e 5,02 MIPS por núcleo que aumentam a velocidade de simulação em cerca de 49 e 147 vezes, respectivamente, em relação a simulação baseada em ISS.

**Conclusões** A principal observação deste estudo de caso está na regularidade dos resultados obtidos e de como um modelo totalmente abstrato de processador é capaz de gerar estimativas razoáveis de tempo simulado. Nas diversas versões de plataformas, com 2 a 16 núcleos, os resultados gerados ilustram que o suporte a preempção do software e captura de eventos de hardware, mesmo com vários núcleos de processamento é consistente.

Caso o modelo de processador utilizado fosse instrumentado com maiores detalhes com relação ao mecanismo de tratamento de interrupção, seria possível observar um resultado extremamente preciso de tempo, uma vez que as estimativas oscilariam sob uma limiar fixo dependente da aplicação. Entretanto, o objetivo aqui é ilustrar o pior cenário possível para a aplicação desta proposta e que seu comportamento é previsível e o erro em algumas aplicações pode ser tolerado.

#### 6.4.3.3 Dhrystone

**Descrição** É um dos mais conhecidas e adotadas aplicações de referência para avaliação de desempenho de processadores, baseando em aritmética de números inteiros e algoritmos sintéticos para aproveitar a capacidade de processamento disponível na plataforma. Cada núcleo de processamento da plataforma executa uma instância desta aplicação para o desempenho de toda a plataforma e de cada núcleo possa ser avaliado.

**Plataforma com 2 núcleos** Nos experimentos com a abordagem proposta realizadas com a aplicação Dhrystone foram utilizados parâmetros de ajuste

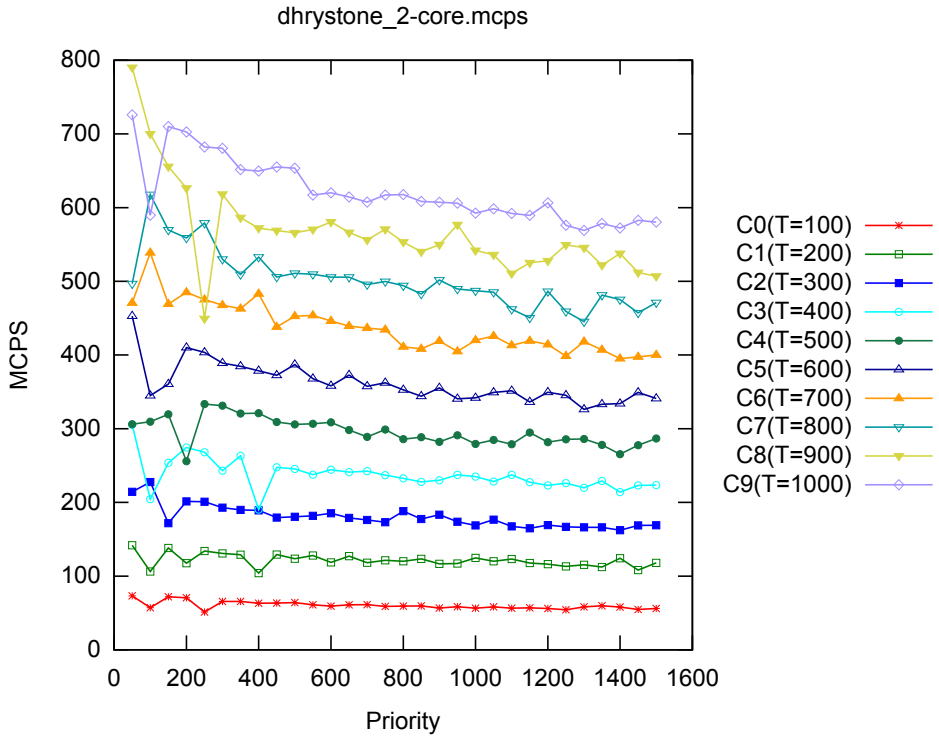


Figura 107: Desempenho em MCPS de Dhrystone (2 núcleos)

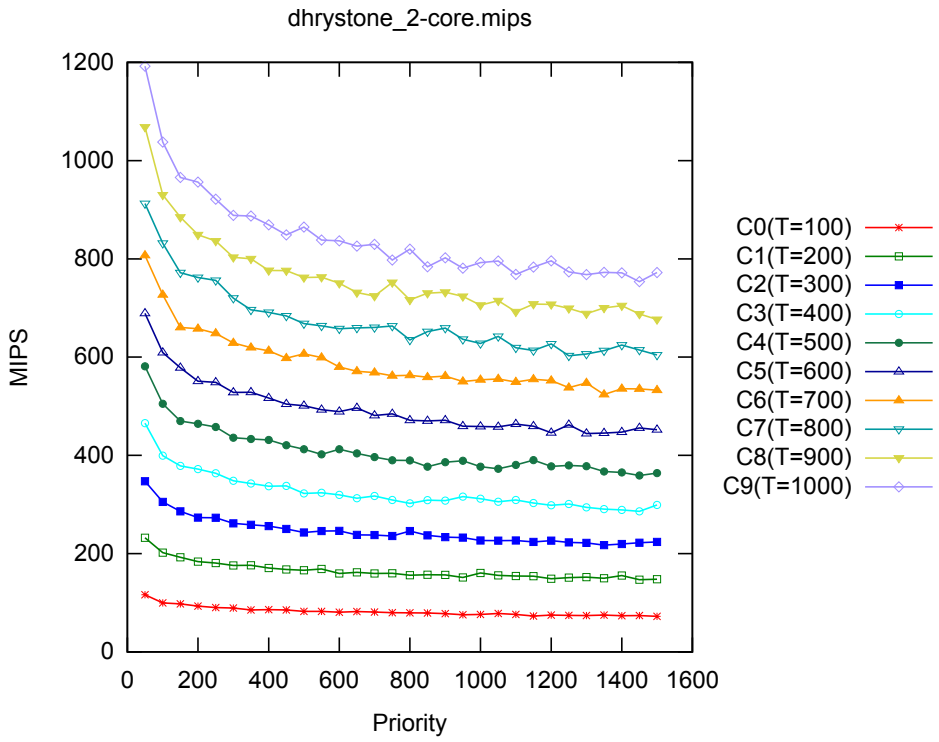


Figura 108: Desempenho em MIPS de Dhrystone (2 núcleos)

com 10 curvas (C0 até C9), de valores entre 100 e 1.000, com intervalos de tamanho 100, e de prioridade com valores entre 50 e 1.500, e passos de tamanho 50. Com esta parametrização foram gerados os gráficos de desempenho vis-

tos nas figuras 107 e 108, que utilizam métricas MCPS e MIPS, respectivamente.

Analisando as curvas de desempenho geradas, pode ser observado um pico de desempenho médio de aproximadamente 800 MCPS e 1.200 MIPS por núcleo que representam um ganho de velocidade de execução da simulação de cerca de 1.379 e 3.750 vezes, respectivamente, em comparação a plataforma que utiliza processadores baseados em ISS com desempenho médio de 0,58 MCPS e 0,32 MIPS.

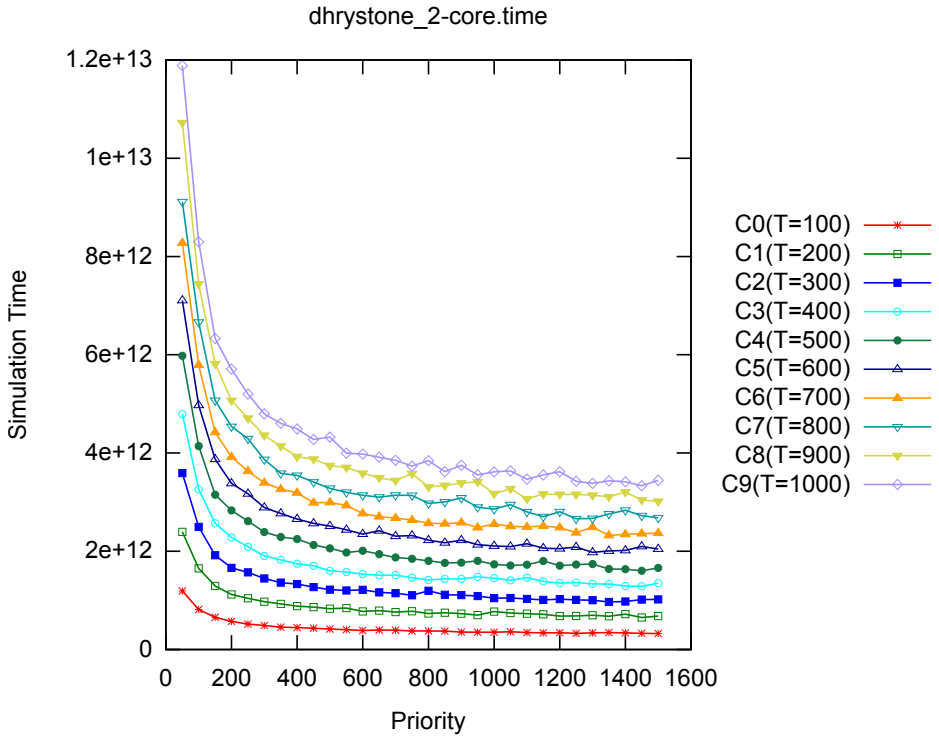


Figura 109: Tempo simulado de Dhrystone (2 núcleos)

Na figura 109 podem ser visualizadas as curvas de estimativa de tempo simulado obtidas utilizando a mesma parametrização. Na simulação em uma plataforma baseada em ISS foi obtido um tempo simulado de 2,218801812e+12 picosegundos, com desempenho médio de 0,58 MCPS e 0,32 MIPS por núcleo de processamento. Para equipar este comportamento no modelo proposto e avaliar o desempenho e erro relativo, foram calibrados parâmetros de ajuste e de prioridade com valores de 599 e 953 que geram uma estimativa de tempo simulado de 2,233235395554e+12 picosegundos em 10 simulações, com erro relativo ao ISS de aproximadamente 0,65% e desvio padrão de 1,3534233724e+10 picosegundos (±0,60%). O desempenho médio do modelo proposto foi de 377,41 MCPS e 507,15 MIPS por núcleo, o que representa um aumento de desempenho de 654 e 1.567 vezes, respectivamente, em compa-

ração aos modelos ISS.

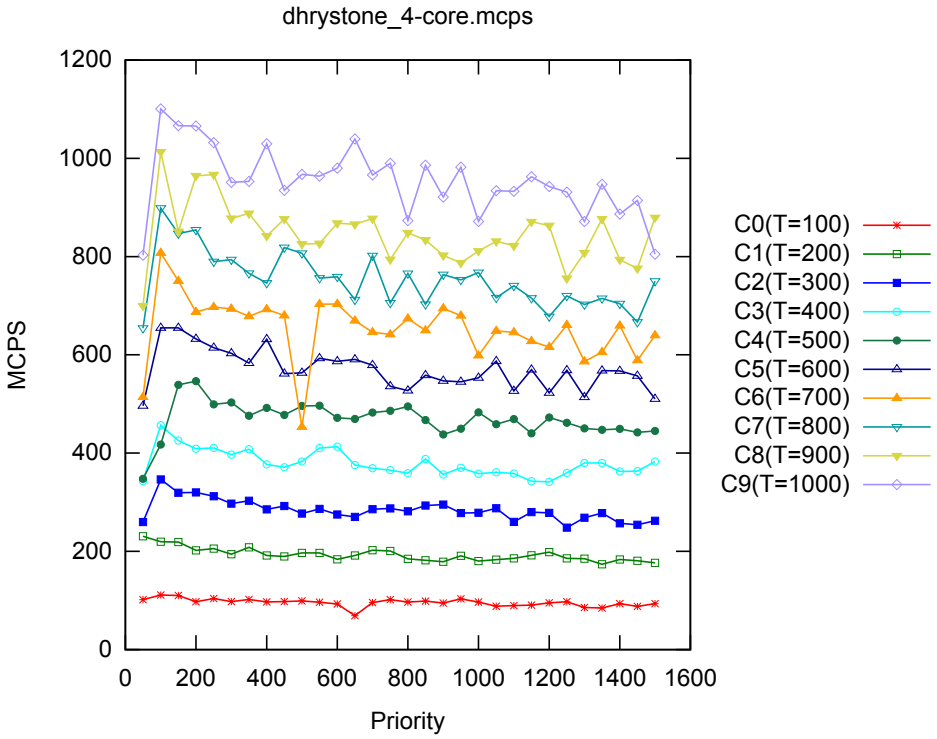


Figura 110: Desempenho em MCPS de Dhrystone (4 núcleos)

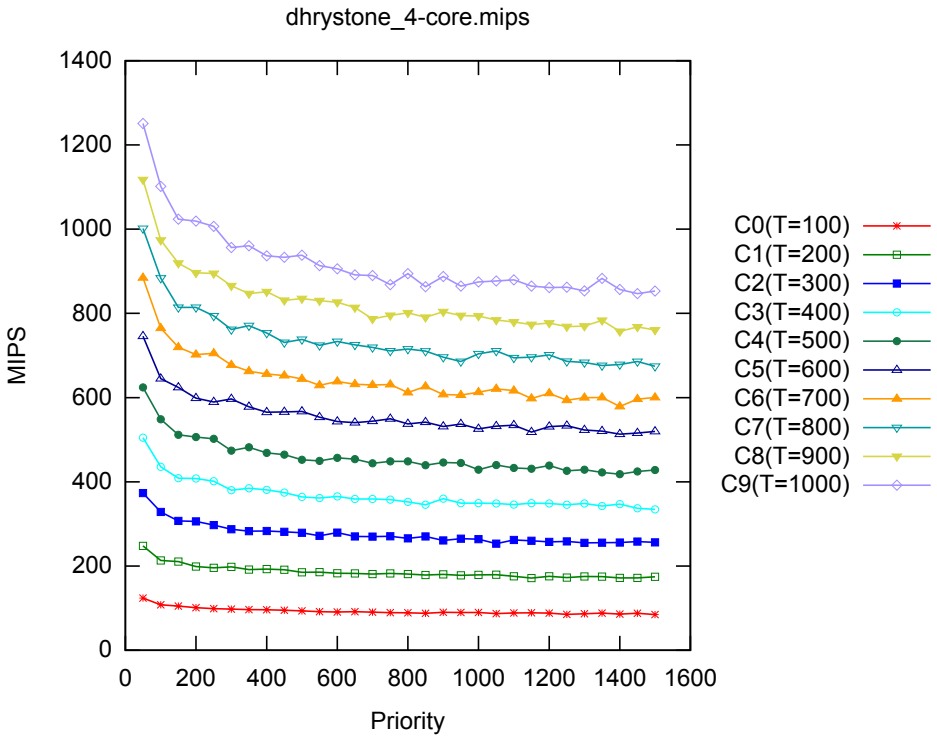


Figura 111: Desempenho em MIPS de Dhrystone (4 núcleos)

**Plataforma com 4 núcleos** Nos experimentos com 4 núcleos de processamento foi utilizada a mesma configuração de parâmetros de ajuste e de prioridade já definidos, gerando os gráficos de desempenho vistos nas figuras 110 e 111, em métricas MCPS e MIPS, respectivamente. Analisando os resultados, pode ser observado um pico de desempenho médio de aproximadamente 1.100 MCPS e 1.200 MIPS por núcleo atingido pelo modelo proposto que representa uma melhoria de cerca de 3.929 e 7.500 vezes, respectivamente, no desempenho em comparação aos modelo ISS que obteve média de 0,28 MCPS e 0,16 MIPS de desempenho por núcleo.

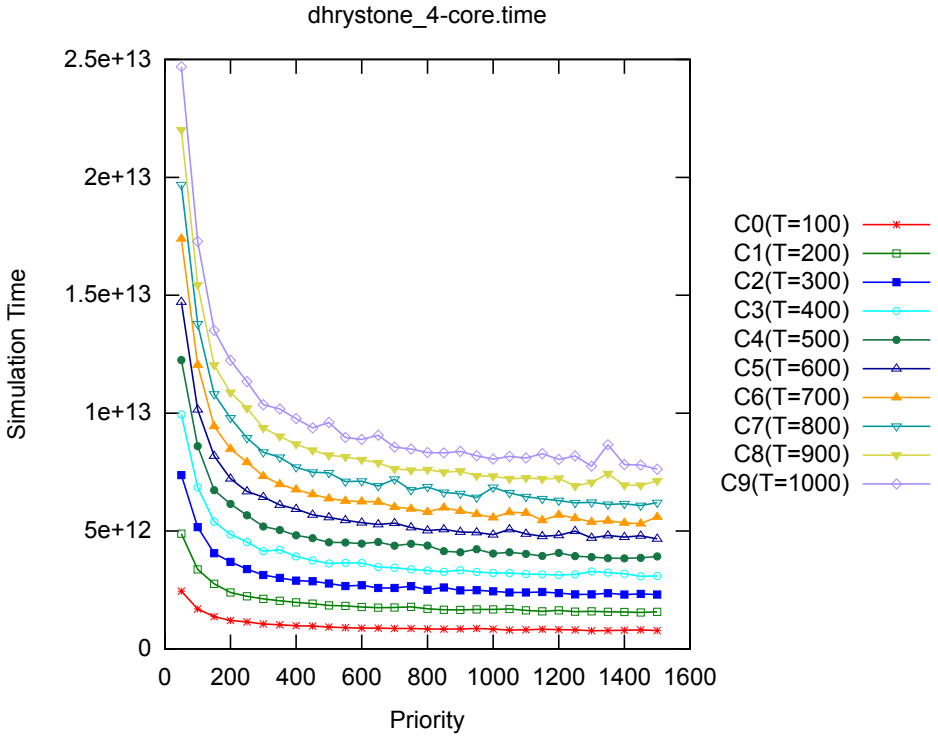


Figura 112: Tempo simulado de Dhrystone (4 núcleos)

Na figura 112 são exibidas as curvas de estimativa de tempo simulado geradas, confirmando mais uma vez o comportamento teórico previsto e a precisão das estimativas estáticas que podem ser realizadas. Na simulação com plataforma baseada em ISS foi atingido um tempo simulado de 2,218838284e+12 picosegundos, com desempenho médio de 0,28 MCPS e 0,16 MIPS por núcleo de processamento. Na configuração do modelo proposto foram calibrados parâmetros de ajuste e de prioridade com valores de 275 e 1.023 para igualar o comportamento observado na plataforma baseada em ISS. A estimativa de tempo simulado obtida foi de 2,233284355294e+12 picosegundos em 5 simulações, com erro relativo ao modelo ISS de cerca de 0,65%, desvio padrão

de 1,2405616902e+10 picosegundos (±0,55%) e desempenho médio de 243,29 MCPS e 236,66 MIPS por núcleo. Estes resultados trazem a conclusão de que o modelo proposto é aproximadamente 860 e 1.492 vezes mais rápido que o modelo ISS, utilizando métricas MCPS e MIPS, respectivamente.

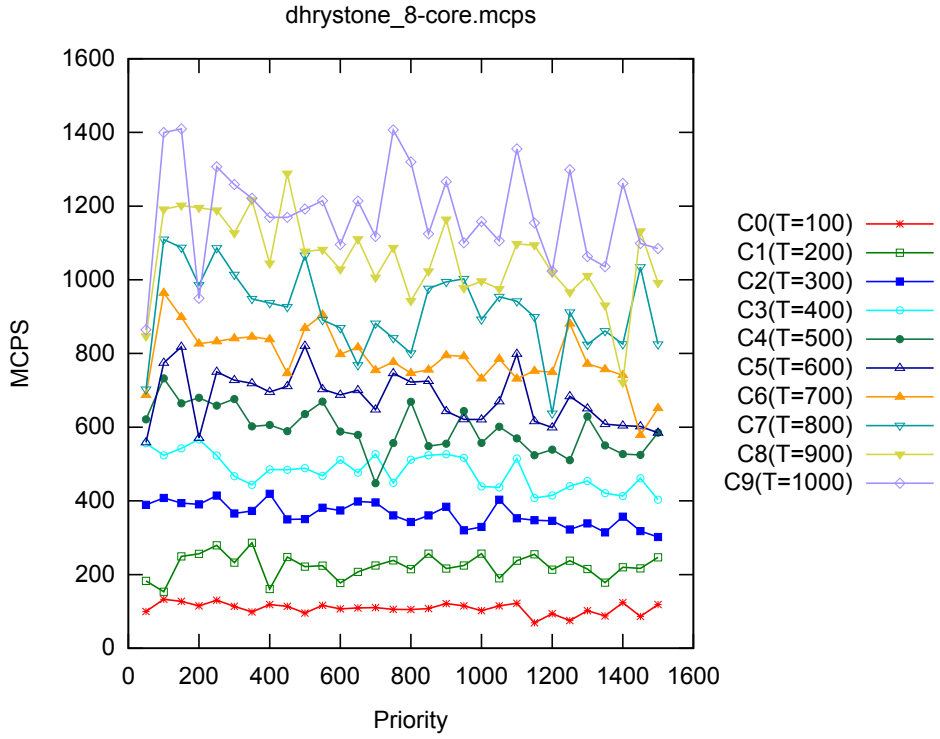


Figura 113: Desempenho em MCPS de Dhrystone (8 núcleos)

**Plataforma com 8 núcleos** Nas figuras 113 e 114 podem ser vistas as curvas de desempenho em MCPS e MIPS geradas, respectivamente, utilizando a configuração de ajuste e de prioridade definidas. Nota-se um pico de desempenho médio de aproximadamente 1.400 MCPS e 1.100 MIPS por núcleo que significa uma melhoria de velocidade de execução de cerca de 10.000 e 13.750 vezes, respectivamente, em comparação ao modelo ISS que obteve 0,14 MCPS e 0,08 MIPS por núcleo.

Observando as curvas de estimativa de tempo simulado na figura 115, pode-se concluir que a aplicação Dhrystone executando em 8 núcleos de processamento também possui o comportamento teórico previsto. Na simulação com plataforma baseada em modelo ISS foi obtido um tempo simulado de 2,218913053e+12, com desempenho médio por núcleo de 0,14 MCPS e 0,08 MIPS. Para configurar o modelo proposto e obter um comportamento equivalente, foram calibrados parâmetros de ajuste e de prioridade com valores de

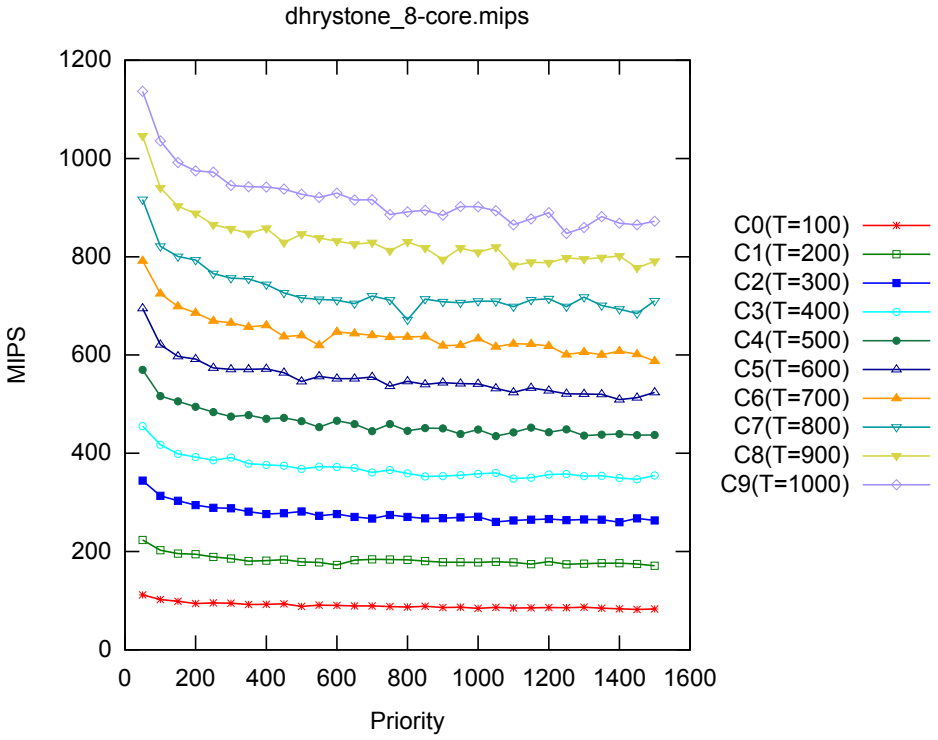


Figura 114: Desempenho em MIPS de Dhrystone (8 núcleos)

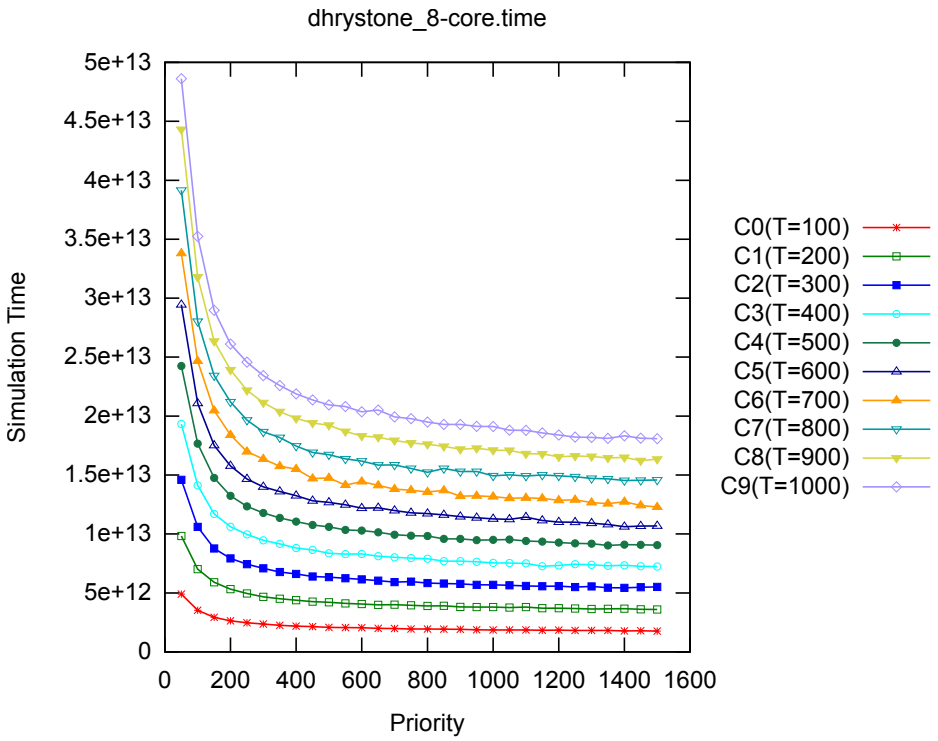


Figura 115: Tempo simulado de Dhrystone (8 núcleos)

119 e 1.073 que geram uma estimativa de tempo simulado de 2,23709613594e+12 picosegundos com 5 simulações. O desempenho médio do modelo proposto foi de 138,42 MCPS e 106,90 MIPS por núcleo que aumenta em cerca de 980

e 1.350 vezes, respectivamente, a velocidade de simulação quando comparado ao modelo ISS, com erro relativo de aproximadamente 0,81% e desvio padrão de 3,0600713218e+10 picosegundos (±1,36%).

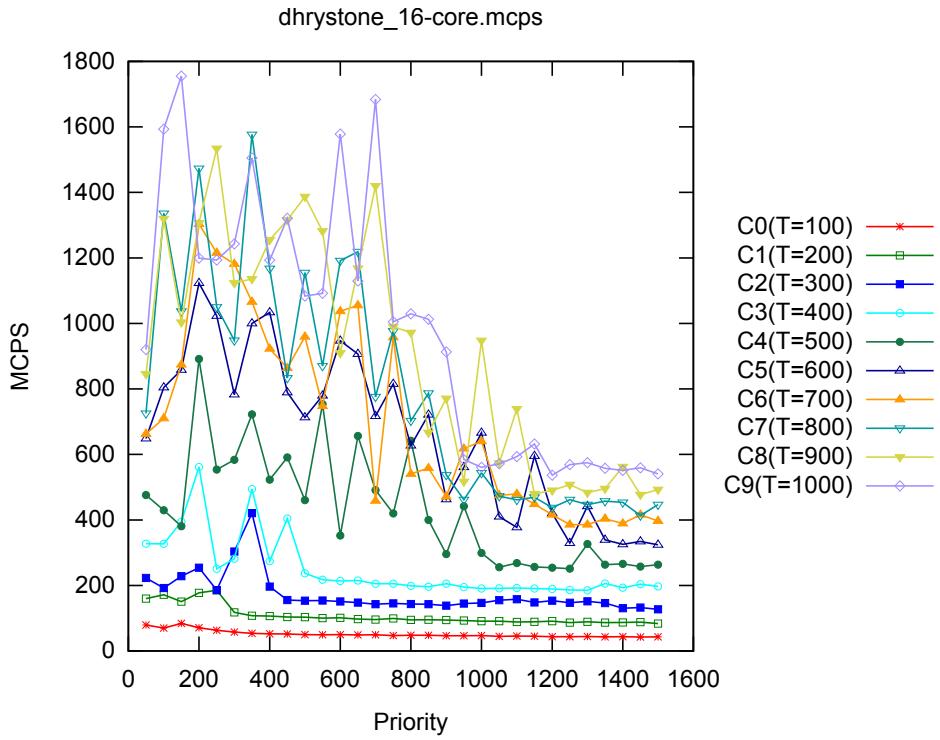


Figura 116: Desempenho em MCPS de Dhrystone (16 núcleos)

**Plataforma com 16 núcleos** Na análise de desempenho do Dhrystone executando em 16 núcleos processamento, adotando a mesma parametrização já definida, são geradas as curvas vistas nas figuras 116 e 117. É possível ver um pico de desempenho médio com cerca de 1.800 MCPS e 1.100 MIPS por núcleo que representam um ganho de velocidade de execução do sistema de aproximadamente 25.714 e 27.500 vezes, respectivamente, quando comparado ao modelo ISS que obteve 0,07 MCPS e 0,04 MIPS de desempenho médio por núcleo.

Na figura 118 são exibidas as estimativas de tempo simulado obtidas durantes os experimentos e pode ser observado um comportamento uniforme nas curvas, com baixo índice de erro e utilizando a parametrização descrita anteriormente. Utilizando a plataforma com processadores baseados em ISS foi obtido um tempo simulado de 2,218985278e+12 picosegundos, com desempenho médio de 0,07 MCPS e 0,04 MIPS por núcleo de processamento. Para fins de análise comparativa, uma plataforma com processadores baseados

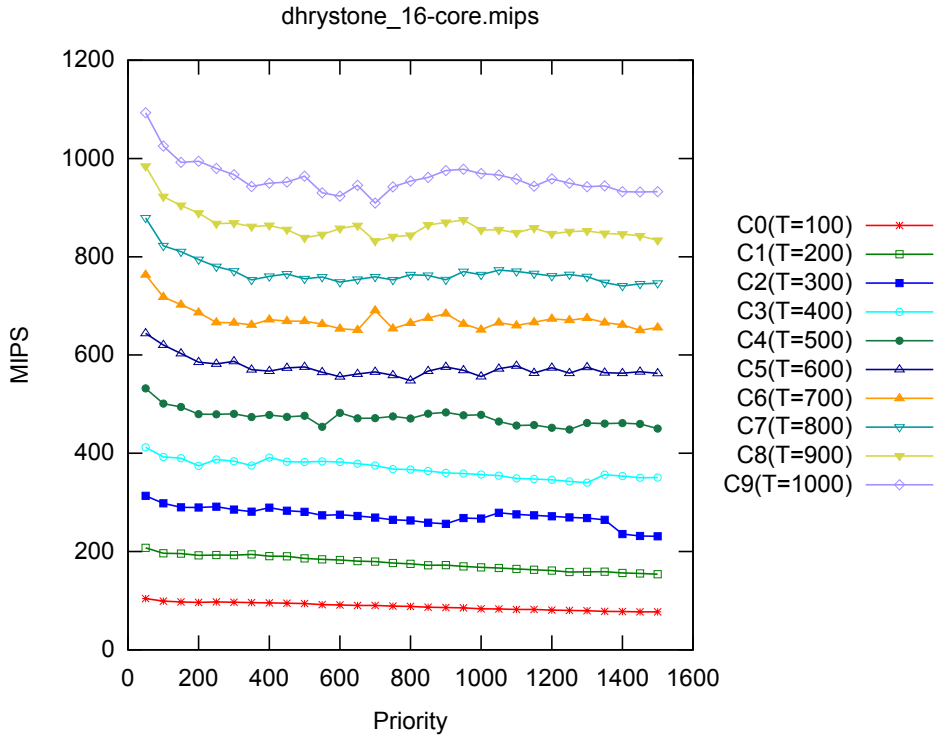


Figura 117: Desempenho em MIPS de Dhrystone (16 núcleos)

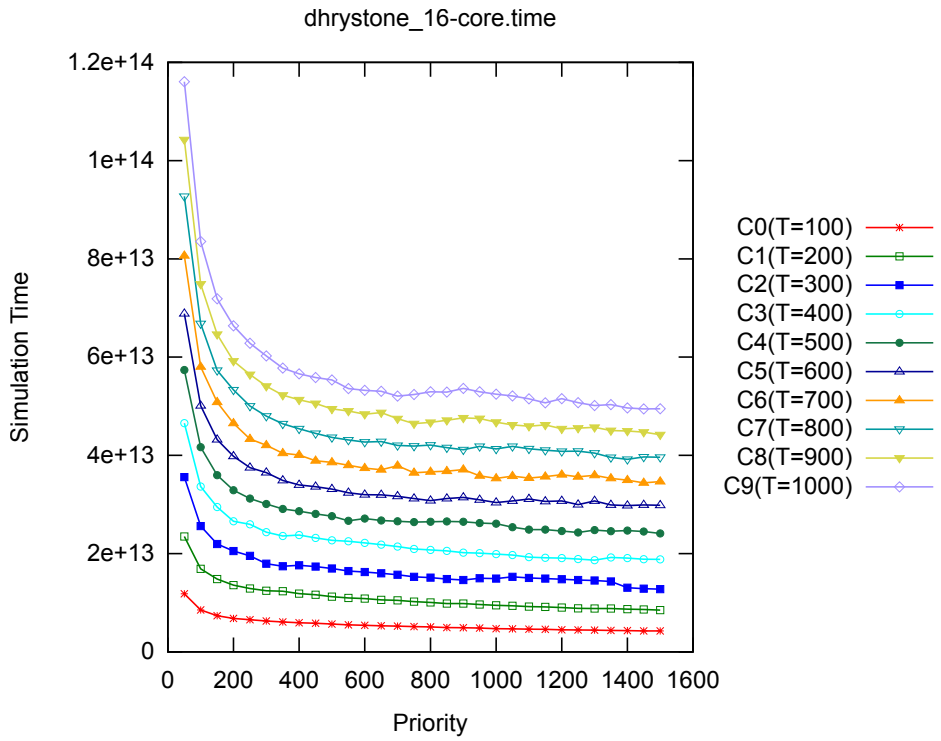


Figura 118: Tempo simulado de Dhrystone (16 núcleos)

no modelo proposto foi configurada com parâmetros de ajuste e de prioridade com valores de 44 e 878 que geram uma estimativa de tempo simulado de 2,194999410604e+12 picosegundos em 4 simulações, com erro relativo ao

ISS de cerca de 1,08% e desvio padrão de 2,1597177779e+10 picosegundos (±0,98%). Os resultados de desempenho médio foram de 38,74 MCPS e 42,03 MIPS por núcleo de processamento que aumentam a velocidade de execução em aproximadamente 553 e 1.070 vezes, respectivamente, em comparação com o modelo ISS.

**Conclusões** Neste estudo de caso a emblemática aplicação Dhrystone foi executada em plataformas de 2 até 16 núcleos, exibindo o comportamento teórico previsto e baixos níveis de erro nas estimativas realizadas. Por ser uma métrica de desempenho bastante difundida, os resultados obtidos permitem ao projetista avaliar o desempenho por núcleo da plataforma criada, sem precisar tomar decisões precoces de projeto, como a escolha da arquitetura de processamento.

Um detalhe importante para ser percebido é que não existem limitações quanto ao número de processadores que podem utilizados na plataforma e os resultados experimentais demonstram que quanto maior for a intensidade computacional ou o número de núcleos de processamento, maior será o número de amostragens realizadas e, consequentemente, melhor será a precisão das estimativas de tempo realizadas.

#### 6.4.3.4 Mergesort

**Descrição** O algoritmo de ordenação Mergesort é um dos melhores exemplos de aplicação paralela que aproveitam completamente os recursos de processamento disponíveis. Baseado em técnicas de divisão e conquista, o passo inicial de sua execução consiste em identificar quantos núcleos de processamento estão disponíveis na plataforma e fazer a alocação de dados que cada um irá ordenar separadamente. Estes dados estão armazenados em uma unidade de armazenamento da plataforma e todos os núcleos concorrem para ordenar o conjunto de dados armazenados.

**Plataforma com 2 núcleos** A realização dos experimentos com a aplicação Mergesort foi feita definindo uma parametrização de ajuste com 10 curvas (C0 até C9), de valores entre 100 e 1.000, com intervalos de tamanho 100, e de pri-

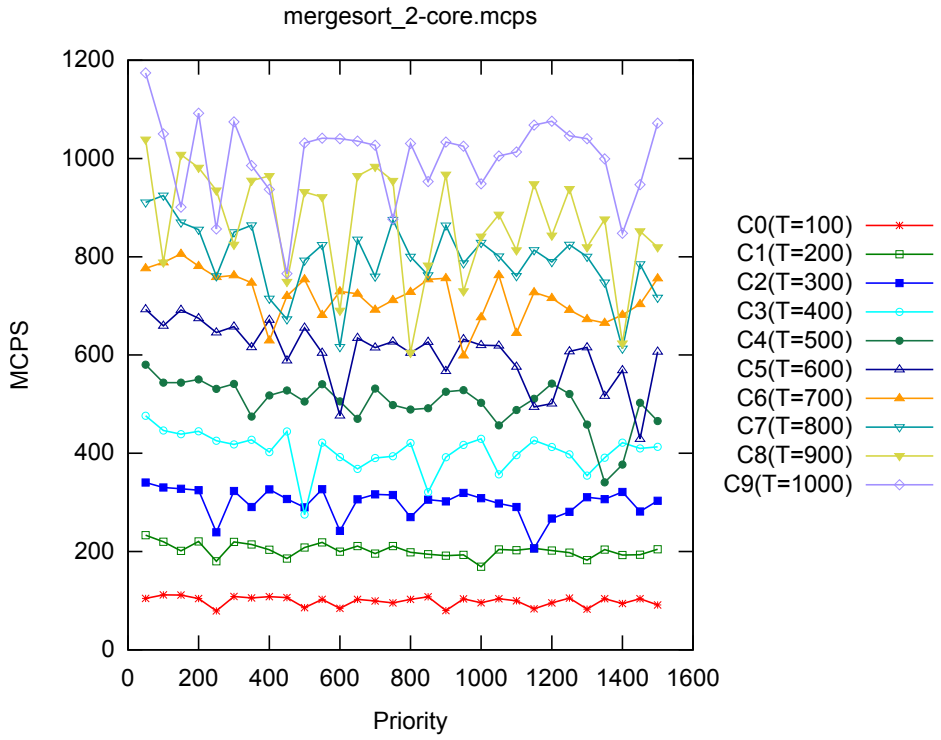


Figura 119: Desempenho em MCPS de Mergesort (2 núcleos)

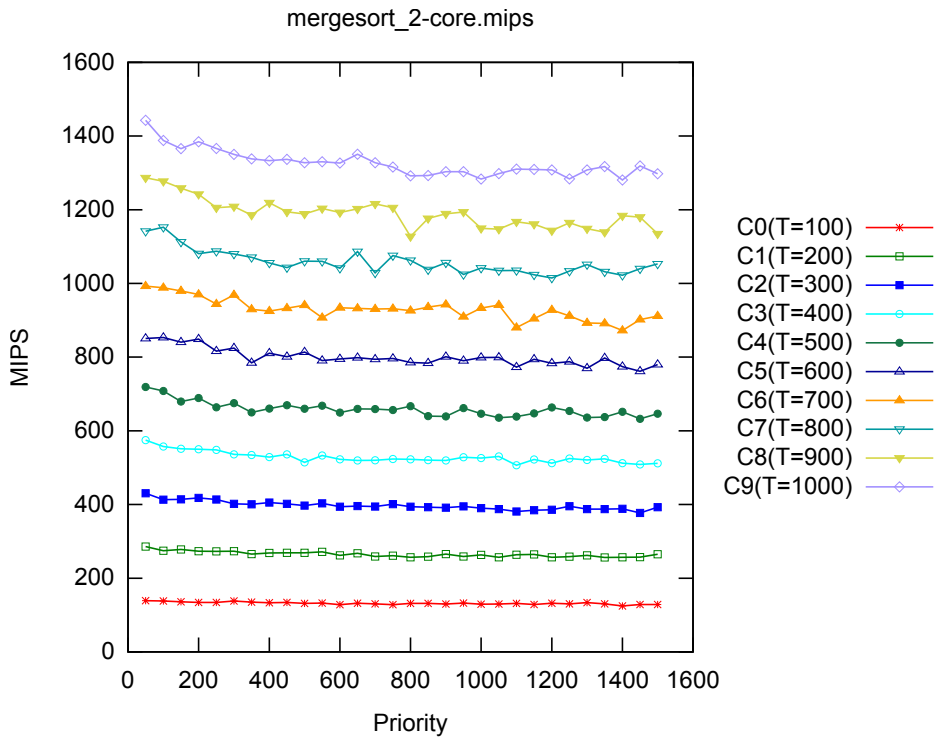


Figura 120: Desempenho em MIPS de Mergesort (2 núcleos)

oridade de valores entre 50 e 1.500, com passos de tamanho 50. Com esta configuração foram gerados os gráficos de desempenho vistos nas figuras 119 e 120, em métricas MCPS e MIPS, respectivamente. Foi atingido pelo modelo

proposto um pico de desempenho médio de aproximadamente 1.200 MCPS e 1.400 MIPS por núcleo, representando um aumento de velocidade de execução de cerca de 522 e 3.590 vezes em comparação com o modelo ISS que obteve desempenho médio de 2,30 MCPS e 0,39 MIPS.

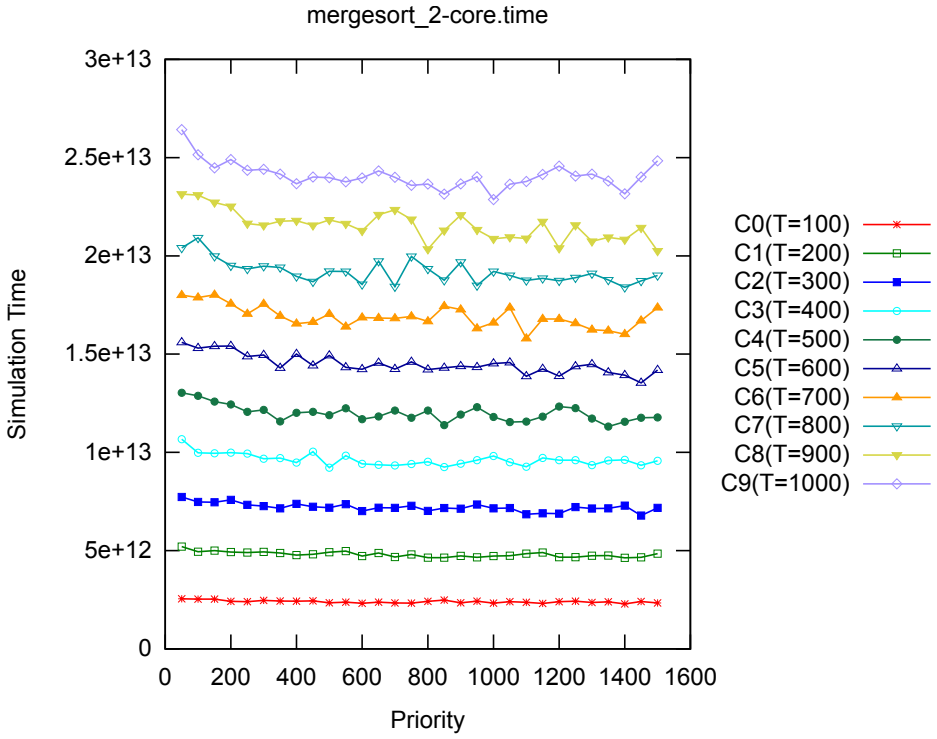


Figura 121: Tempo simulado de Mergesort (2 núcleos)

As estimativas de tempo geradas podem ser vistas na figura 121, considerando a parametrização definida. Pode ser observar o mesmo comportamento descrito pelo modelo teórico, apresentando uma taxa de variação mais suave do que nas outras aplicações. Na simulação com plataforma baseada em ISS foi obtido um tempo simulado de 4,25718851e+11 picosegundos, com desempenho médio por núcleo de 2,30 MCPS e 0,39 MIPS. Para equiparar este comportamento no modelo proposto, foram calibrados parâmetros de ajuste e de prioridade com valores de 20 e 679 que geram uma estimativa de tempo simulado de 4,43129259059e+11 picosegundos em 8 simulações, com erro relativo ao ISS de cerca de 4,08% e desvio padrão de 4,304551195e+9 picosegundos (±0,97%). Os resultados também mostram um desempenho médio de 16,25 MCPS e 24,02 MIPS que tornam o modelo proposto cerca de 7 e 61 vezes mais rápido do que o modelo ISS.

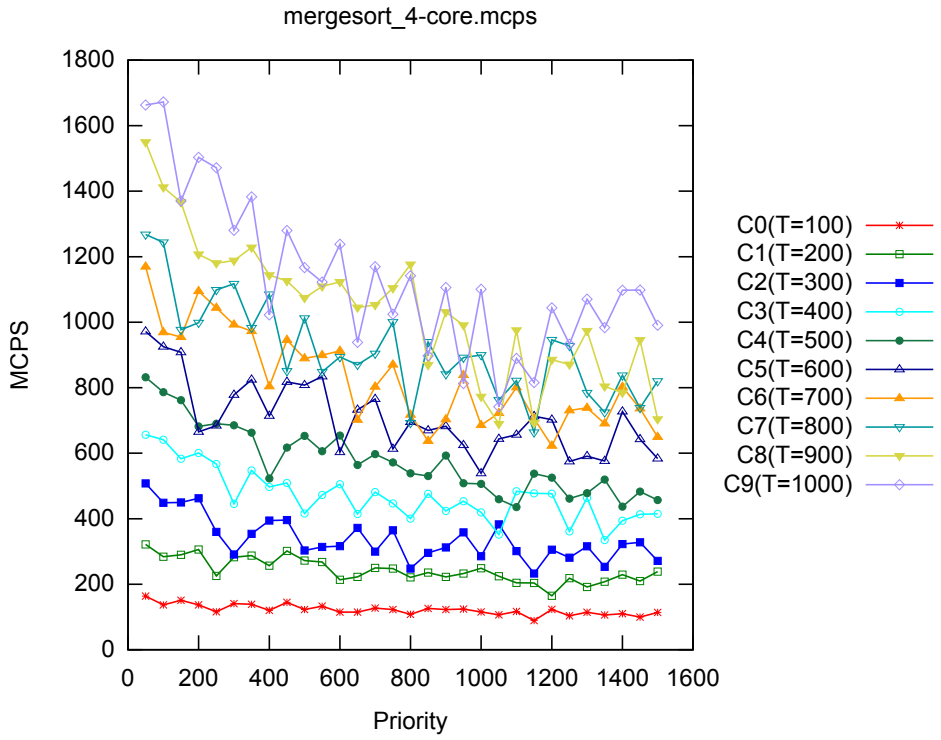


Figura 122: Desempenho em MCPS de Mergesort (4 núcleos)

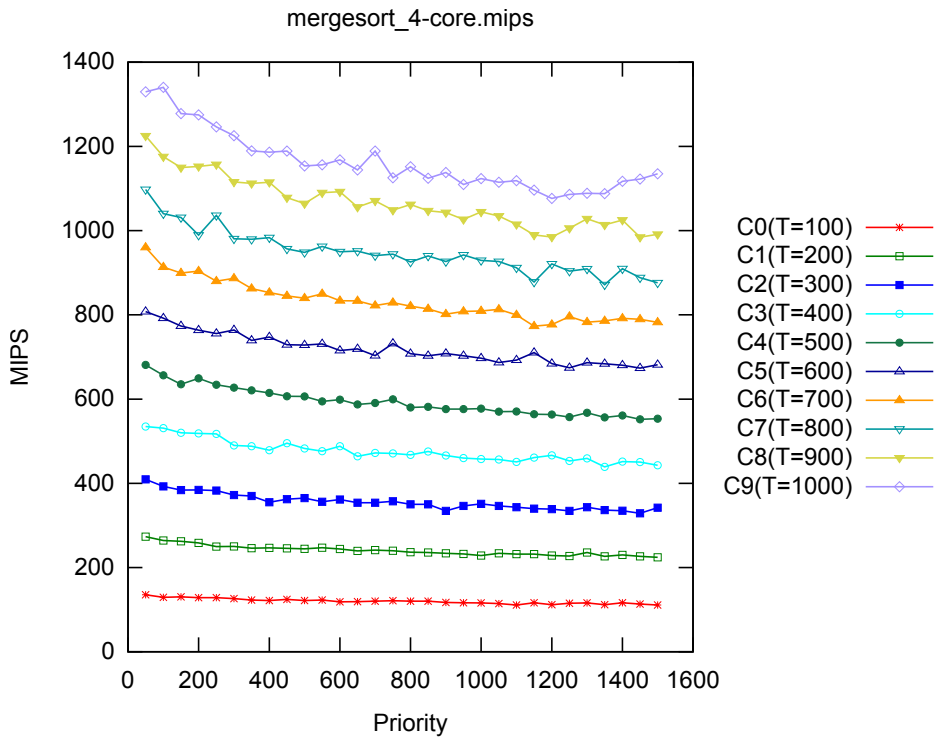


Figura 123: Desempenho em MIPS de Mergesort (4 núcleos)

**Plataforma com 4 núcleos** Para geração dos gráficos de desempenho vistos nas figuras 122 e 123 são utilizados os mesmos valores de parâmetros de ajuste e de prioridade definidos anteriormente. Pode-se observar um pico de

desempenho médio de aproximadamente 1.700 MCPS e 1.300 MIPS por núcleo, resultado que demonstra que o modelo proposto é cerca de 614 e 6.191 vezes, respectivamente, mais rápido que o modelo ISS que obteve 2,77 MCPS e 0,21 MIPS de desempenho médio por núcleo.

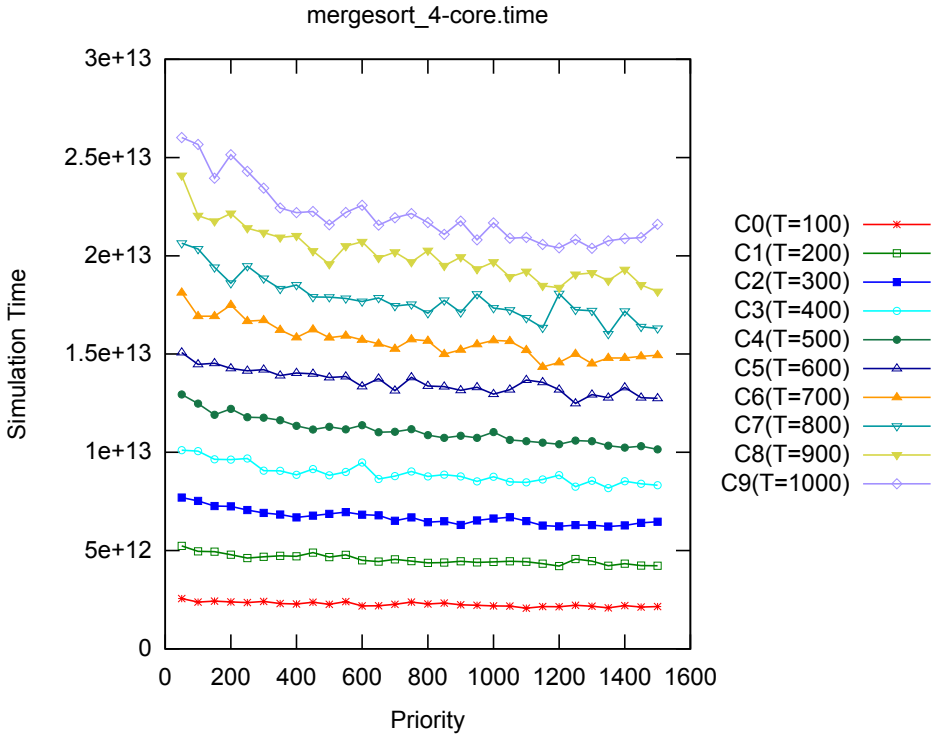


Figura 124: Tempo simulado de Mergesort (4 núcleos)

Na figura 124 são exibidas as estimativas de tempo simulado para o Mergesort executando em 4 processadores, utilizando a mesma parametrização definida nos gráficos anteriores. Para obter informações de referência foi utilizada a plataforma baseada em ISS que gerou um tempo simulado de 4,22066934e+11 picosegundos, com desempenho médio de 2,77 MCPS e 0,21 MIPS por núcleo de processamento. A configuração do modelo proposto busca obter o mesmo comportamento observado na plataforma de referência e para isto foram calibrados parâmetros de ajuste e de prioridade com valores de 19 e 564 que geram uma estimativa de tempo simulado de 4,19619903088e+11 picosegundos em 20 simulações, com erro relativo de cerca de 0,57% e desvio padrão de 5,212532775e+9 picosegundos (±1,24%). O desempenho do modelo proposto foi de 19,92 MCPS e 21,87 MIPS que o torna cerca de 7 e 103 vezes mais rápido que o modelo ISS.

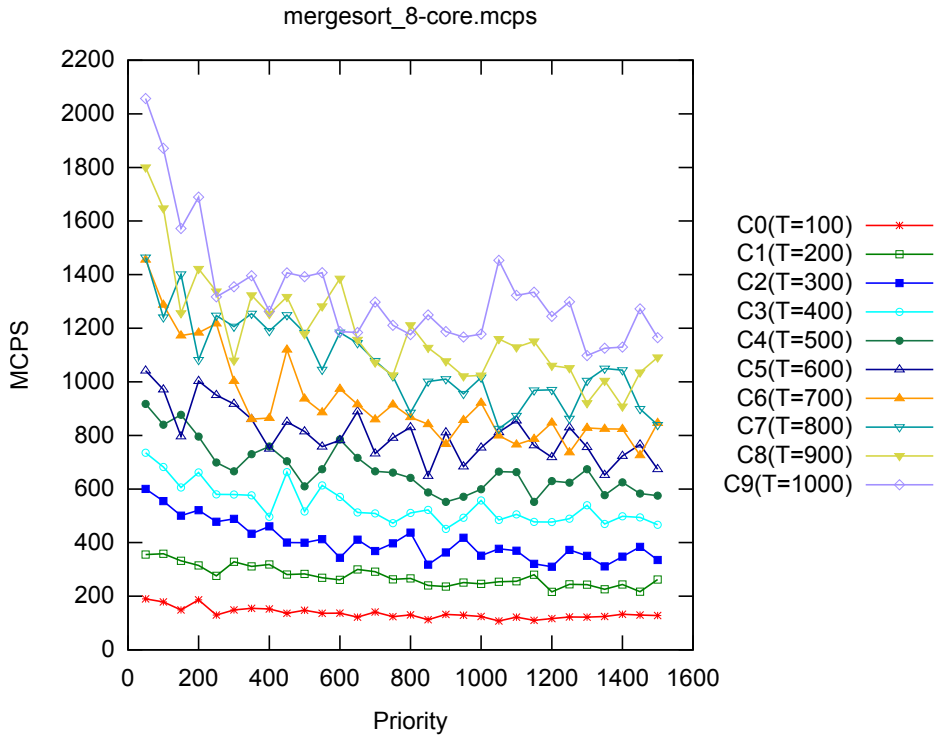


Figura 125: Desempenho em MCPS de Mergesort (8 núcleos)

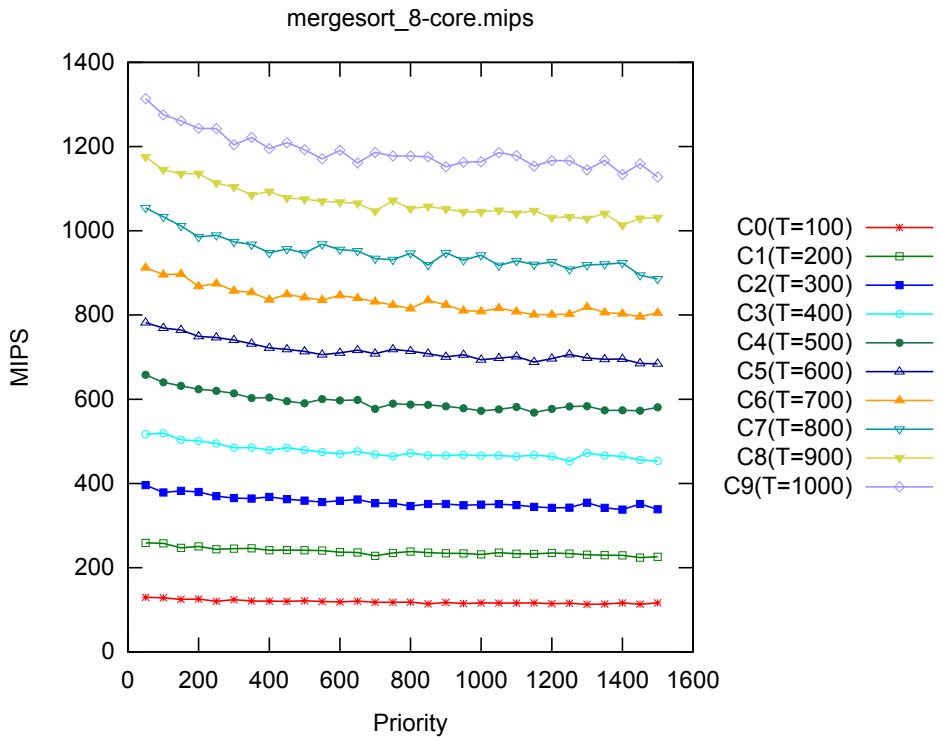


Figura 126: Desempenho em MIPS de Mergesort (8 núcleos)

**Plataforma com 8 núcleos** A análise de desempenho do Mergesort pode ser visualizada nas figuras 125 e 126 que traçam as curvas obtidas em métricas MCPS e MIPS, respectivamente. É possível ver um pico de desempenho médio

de cerca de 2.000 MCPS e 1.300 MIPS por cada núcleo de processamento, resultado que demonstra que o modelo proposto é aproximadamente 604 e 10.833 vezes mais rápido do que o modelo ISS que obteve 3,31 MCPS e 0,12 MIPS de desempenho médio por núcleo.

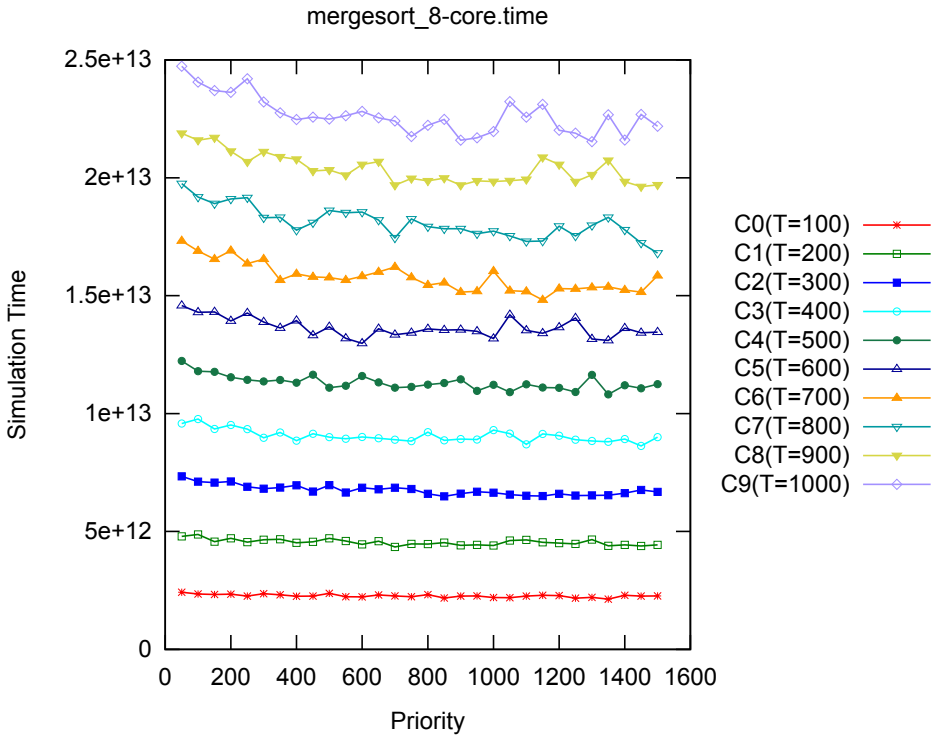


Figura 127: Tempo simulado de Mergesort (8 núcleos)

Na figura 127 podem ser vistas as estimativas de tempo simulado geradas durante os experimentos realizados com o Mergesort. Os resultados obtidos com a simulação baseada em ISS mostram um tempo simulado de 4,32256556e+11 picosegundos, com desempenho médio de cada processador de 3,31 MCPS e 0,12 MIPS. Durante a configuração da plataforma empregando a abordagem proposta foram calibrados parâmetros de ajuste e de prioridade com valores de 19 e 989 que geram uma estimativa de tempo simulado de 4,20224939309e+11 picosegundos em 12 simulações, com erro relativo ao ISS de 2,78% e desvio padrão de 5,397337985e+9 picosegundos (±1,28%). O modelo proposto atingiu um desempenho médio de 21,76 MCPS e 21,14 MIPS por processador, aumentando a velocidade de execução em cerca de 7 e 183 vezes, respectivamente, em comparação com o modelo ISS.

**Plataforma com 16 núcleos** Para a realização da análise de desempenho médio obtido por cada processador, utilizando as métricas MCPS e MIPS, fo-

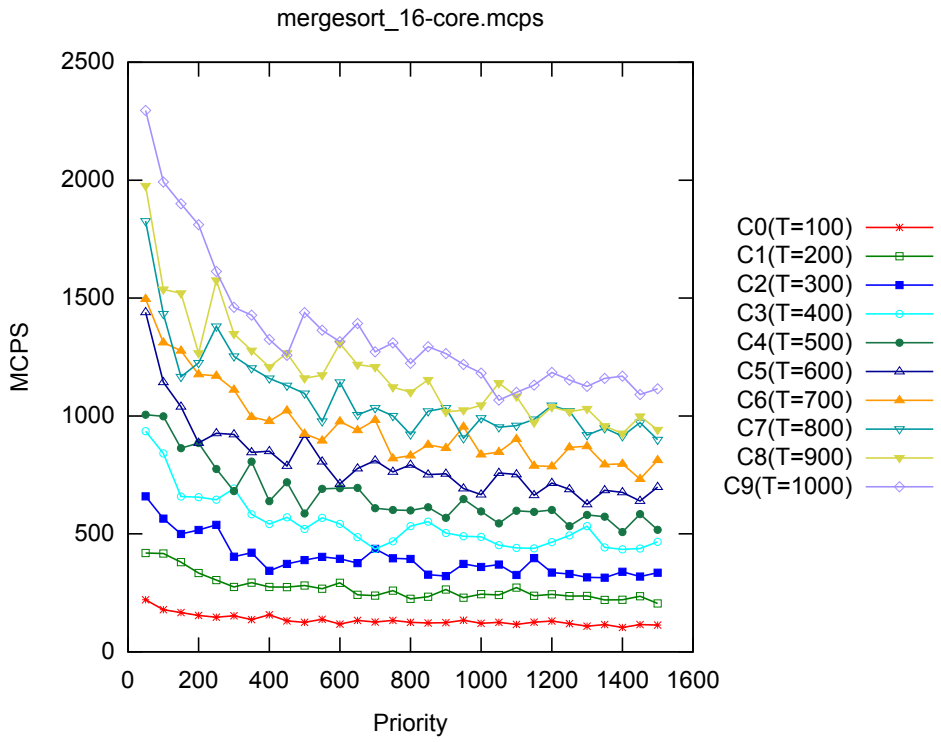


Figura 128: Desempenho em MCPS de Mergesort (16 núcleos)

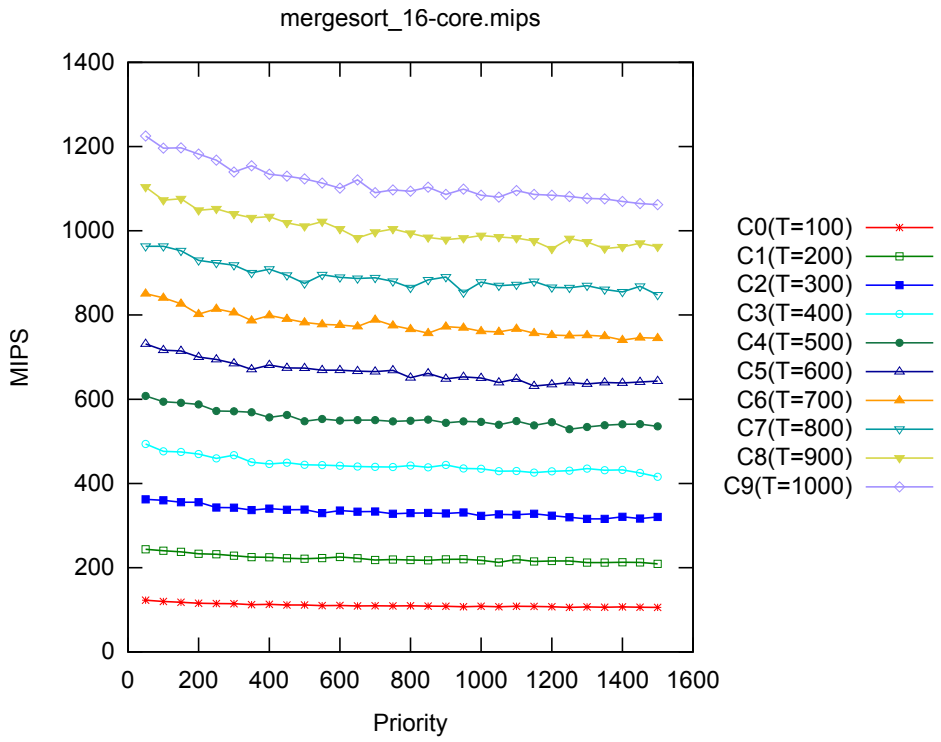


Figura 129: Desempenho em MIPS de Mergesort (16 núcleos)

ram gerados os gráficos vistos nas figuras 128 e 129. É possível ver um pico de desempenho médio com aproximadamente 2.250 MCPS e 1.200 MIPS por processador, o que proporciona um aumento de velocidade de execução de

cerca de 575 e 20.000 vezes, respectivamente, com relação ao desempenho médio do ISS que foi de 3,91 MCPS e 0,06 MIPS por núcleo de processamento.

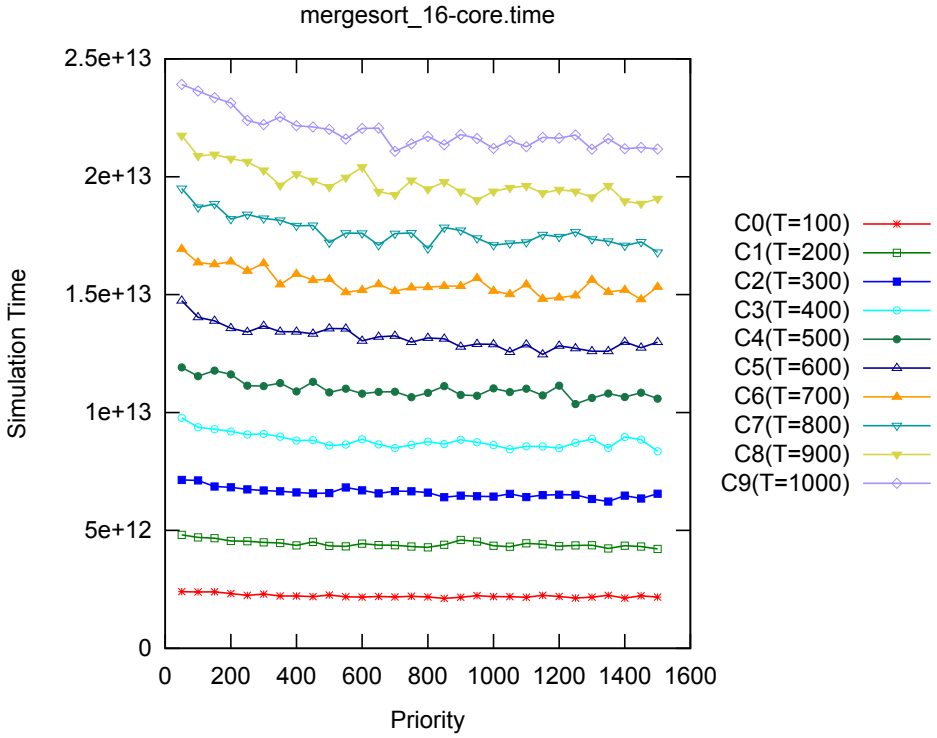


Figura 130: Tempo simulado de Mergesort (16 núcleos)

O comportamento das curvas de estimativas de tempo simulado realizadas podem ser vistas na figura 130, utilizando a mesma parametrização já definida e aplicada na geração dos gráficos anteriores. A simulação com a plataforma baseada em ISS atingiu um tempo simulado de 4,42545736e+11 picosegundos e o desempenho médio de cada processador foi de 3,91 MCPS e 0,06 MIPS. Para configurar o modelo proposto e obter um comportamento mais próximo possível, foram definidos parâmetros de ajuste e de prioridade com valores de 19 e 1.148 que geram uma estimativa de tempo simulado de 4,32886255845e+11 picosegundos com 17 simulações. Este resultado possui um erro relativo de 2,18%, um desvio padrão de 8,184003284e+9 picosegundos (±1,89%) e apresenta desempenho médio de 22,88 MCPS e 20,47 MIPS, tornando a abordagem proposta cerca de 6 e 333 vezes mais rápida que a baseada em ISS.

**Conclusões** O estudo de caso da aplicação Mergesort é um dos mais interessantes exemplos utilizados e isto se deve a sua concorrência de execução no acesso ao dispositivo de armazenamento com tempo de acesso aleató-

rio. Estas duas características conferem a este estudo de caso importantes características que colocam a prova o modelo de tempo criado e confirmam que seu comportamento se mantem mesmo em cenários não comportados de uso.

Com relação ao modelo de tempo, em todos as plataformas criadas com 2 até 16 processadores executam a tarefa de ordenação concorrendo pelo acesso ao dado armazenado na unidade de memória flash do barramento. Mesmo com o acesso de tempo aleatório ao dispositivo e ao barramento pelos processadores, as estimativas de tempo apresentam a característica prevista no modelo teórico. A expectativa é que este estudo de caso demonstre que o modelo proposto de sistema é simples, eficiente e capaz de desempenhar o seu papel em contextos de multiprocessamento com comunicação entre os núcleos, mesmo com dispositivos com tempo de acesso aleatório e com estimativas de tempo simulado com excelente precisão.

#### 6.4.3.5 Análise Comparativa

Esta seção é dedicada a sistematizar todos os dados experimentais coletados durante as simulações multiprocessadas, fornecendo uma visão centrada no número de núcleos de processamento utilizados. Serão colocados os dados de desempenho da abordagem proposta (HdSC) e do modelo de referência (ISS), utilizando as métricas MCPS e MIPS, respectivamente. Com estas informações será estabelecido um valor para ganho de desempenho em cada métrica e o erro associado ao modelo proposto com relação ao modelo ISS.

Tabela 22: Comparativo de aplicações multiprocessadas (2 núcleos)

| Aplicação | HdSC | ISS | Desempenho | Erro |
|---|---|---|---|---|
| CoreMark | 257,10 MCPS<br>344,30 MIPS | 0,53 MCPS<br>0,32 MIPS | 489x<br>1.073x | 0,77% |
| Device Control | 45,94 MCPS<br>67,54 MIPS | 0,44 MCPS<br>0,15 MIPS | 115x<br>360x | 10,55% |
| Dhrystone | 377,41 MCPS<br>507,15 MIPS | 0,58 MCPS<br>0,32 MIPS | 654x<br>1.567x | 0,65% |
| Mergesort | 16,25 MCPS<br>24,02 MIPS | 2,30 MCPS<br>0,39 MIPS | 7x<br>61x | 4,08% |

Na tabela 22 podem ser observados os dados experimentais obtidos du-

rante as simulações com plataformas com 2 núcleos de processamento. Os resultados possuem um desempenho médio de cerca de 174,18 MCPS e 235,75 MIPS para a abordagem proposta HdSC, desempenho médio de cerca de 0,77 MCPS e 0,24 MIPS para o modelo ISS, com ganhos de desempenho relativo de aproximadamente 226 e 982 vezes, respectivamente, e com erro médio de 4,01%.

Tabela 23: Comparativo de aplicações multiprocessadas (4 núcleos)

| Aplicação | HdSC | ISS | Desempenho | Erro |
|---|---|---|---|---|
| CoreMark | 165,99 MCPS<br>157,50 MIPS | 0,25 MCPS<br>0,15 MIPS | 665x<br>1.034x | 0,53% |
| Device Control | 20,12 MCPS<br>34,03 MIPS | 0,19 MCPS<br>0,12 MIPS | 104x<br>317x | 8,18% |
| Dhrystone | 243,29 MCPS<br>236,66 MIPS | 0,28 MCPS<br>0,16 MIPS | 860x<br>1.492x | 0,65% |
| Mergesort | 19,92 MCPS<br>21,87 MIPS | 2,77 MCPS<br>0,21 MIPS | 7x<br>103x | 0,57% |

Na tabela 23 são exibidas as informações de simulação obtidas durante os experimentos com plataformas com 4 núcleos de processamento. O desempenho médio do modelo proposto (HdSC) foi de cerca de 112,33 MCPS e 112,52 MIPS por núcleo, e do modelo ISS foi de aproximadamente 0,70 MCPS e 0,13 MIPS. Estes resultados trazem um ganho de desempenho de 161 e 866 vezes, respectivamente, com erro médio de 2,48%.

Tabela 24: Comparativo de aplicações multiprocessadas (8 núcleos)

| Aplicação | HdSC | ISS | Desempenho | Erro |
|---|---|---|---|---|
| CoreMark | 96,11 MCPS<br>72,45 MIPS | 0,13 MCPS<br>0,08 MIPS | 756x<br>934x | 0,63% |
| Device Control | 7,08 MCPS<br>11,85 MIPS | 0,12 MCPS<br>0,06 MIPS | 58x<br>184x | 6,54% |
| Dhrystone | 138,42 MCPS<br>106,90 MIPS | 0,14 MCPS<br>0,08 MIPS | 980x<br>1.350x | 0,81% |
| Mergesort | 21,76 MCPS<br>21,14 MIPS | 3,31 MCPS<br>0,12 MIPS | 7x<br>183x | 2,78% |

São fornecidos na tabela 24 todas as informações coletadas durante os experimentos com plataformas com 8 núcleos de processamento. Os resultados mostram um desempenho médio de 65,84 MCPS e 53,09 MIPS para a abordagem proposta (HdSC), e de 0,74 MCPS e 0,07 MIPS para o modelo de

referência (ISS). A melhoria de desempenho médio obtido foi de 89 e 758 vezes, respectivamente, com erro médio de 2,69%.

Tabela 25: Comparativo de aplicações multiprocessadas (16 núcleos)

| Aplicação | HdSC | ISS | Desempenho | Erro |
|---|---|---|---|---|
| CoreMark | 37,87 MCPS<br>28,72 MIPS | 0,06 MCPS<br>0,04 MIPS | 608x<br>756x | 1,13% |
| Device Control | 3,15 MCPS<br>5,02 MIPS | 0,06 MCPS<br>0,03 MIPS | 49x<br>147x | 4,00% |
| Dhrystone | 38,74 MCPS<br>42,03 MIPS | 0,07 MCPS<br>0,04 MIPS | 553x<br>1.070x | 1,08% |
| Mergesort | 22,87 MCPS<br>20,47 MIPS | 3,91 MCPS<br>0,06 MIPS | 6x<br>333x | 2,18% |

Na tabela 25 são sistematizados os dados de desempenho e de erro coletados durante os experimentos realizados em plataformas com 16 núcleos de processamento. Os resultados mostram um desempenho médio de 25,66 MCPS e 24,06 MIPS para a abordagem proposta (HdSC), e de 0,82 MCPS e 0,03 MIPS para a plataforma de referência (ISS). A melhoria de média desempenho gerada foi de 31 e 802 vezes, respectivamente, com erro médio de 2,10%.

## 6.5 Análise das Estimativas de Desempenho

Na seção anterior foram detalhadas as principais aplicações utilizadas nos estudos de caso, contemplando três categorias de aplicação: entrada e saída intensiva, computação intensiva e multiprocessada. Esta seção é dedicada a fornecer uma análise comparativa mais sistemática entre as aplicações que contemplem todas as categorias definidas nos estudas de caso, permitindo ao leitor avaliar a consistência e fidelidade das estimativas geradas pela abordagem proposta com relação as características de desempenho em plataformas, sob a perspectiva da otimização de código e da execução multiprocessada. Para realização de todas as análises das estimativas de desempenho desta seção foram utilizados os dados estatísticos coletados durante a realização dos estudos de caso, de acordo com as definições realizadas nas seções 6.2 e 6.3 deste capítulo que tratam sobre a metodologia e as métricas adotadas.

As aplicações escolhidas para as análises experimentais foi o CoreMark,

por ser da categoria de computação intensiva, amplamente difundido e com resultados mensuráveis, e o Mergesort, por realizar entrada e saída intensiva em sua execução, podendo ter sua execução paralelizada entre os núcleos de processamento da plataforma. Além destas características já mencionadas, estas aplicações executam em todas as configurações de plataforma desenvolvidas nos estudos de caso, permitindo uma análise representativa das estimativas de desempenho geradas.

### 6.5.1 Impacto da Otimização de Código

Como foi esclarecido na seção 6.2 de definição de metodologia dos experimentos, todas as simulações realizadas utilizaram as configurações padronizadas para geração de código, sem otimização para área ou para desempenho. O objetivo desta subseção é investigar o impacto da otimização de código (diretiva -O3) feita pelo compilador nas estimativas de tempo realizadas, através da análise do comportamento das curvas de estimativa de tempo simulado geradas e dos parâmetros de configuração do sistema.

#### 6.5.1.1 CoreMark

A aplicação CoreMark é caracterizada pela utilização de algoritmos reais utilizados em diversas aplicações industriais e por calcular no final de sua execução uma pontuação que permite a comparação de desempenho com outras plataformas. Nesta análise em particular, foram realizadas diversas simulações com o objetivo de comparar os efeitos da otimização de código pelo compilador no comportamento e no desempenho de simulação obtidos.

Apesar da alta resolução das estimativas de tempo simulado em picosegundos, como pode ser visto na figura 131, é possível observar que existe um padrão de estimativas com otimização (parte inferior) que possui valores inferiores em relação à compilação sem otimização (parte superior). Isto pode ser explicado pela redução de tempo de execução nativo dos blocos básicos da aplicação, o que vai implicar em uma estimativa de tempo simulado também menor.

Nas figuras 132 e 133 são comparados os desempenhos em MCPS e MIPS, respectivamente, onde a parte superior de cada figura representa os dados

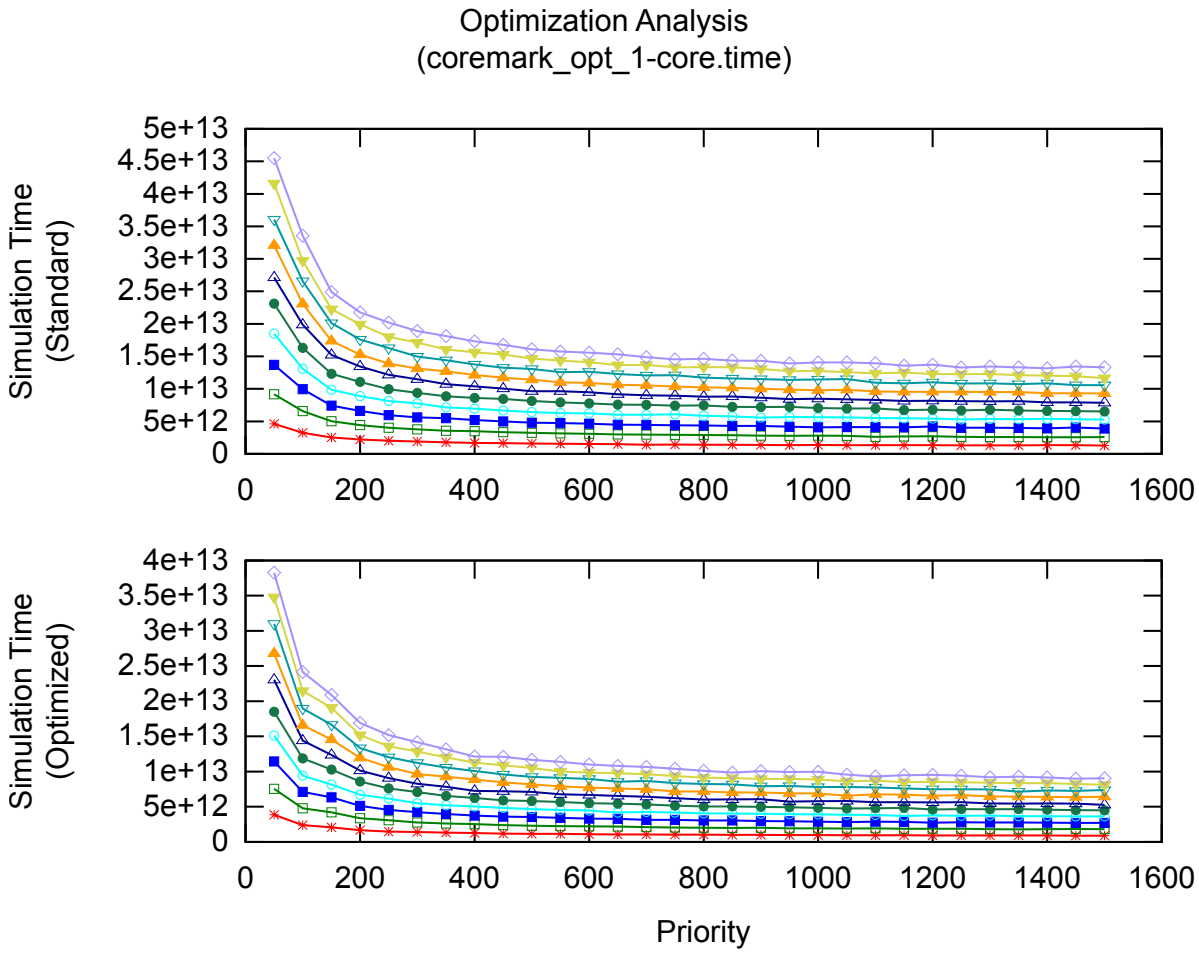


Figura 131: Comparativo de estimativa de tempo simulado do CoreMark

coletados a execução do código sem otimizações e a parte inferior apresenta os dados obtidos de simulações onde a aplicação foi otimizada pelo compilador. Devido a escala das figuras, não é possível verificar facilmente a diferença de desempenho entre as simulações, por isso a comparação será realizada através dos parâmetros de configuração do sistema.

A calibração do CoreMark definiu os parâmetros de ajuste e de prioridade com valores de 837 e 1.058, respectivamente, gerando um desempenho de simulação de aproximadamente 360 MCPS e 721 MIPS. Mantendo o valor do parâmetro de prioridade, é feita a calibração de um novo valor de ajuste para realizar a comparação entre a diferença de desempenho das plataformas com e sem a utilização de otimização de código. Após a conclusão da calibração, foi obtido um parâmetro de ajuste com valor 1221 que apresentou um desempenho de simulação de cerca de 498 MCPS e 995 MIPS. Este resultado representam uma melhoria de aproximadamente 38% no desempenho da simulação, mantendo as mesmas características funcionais da aplicação

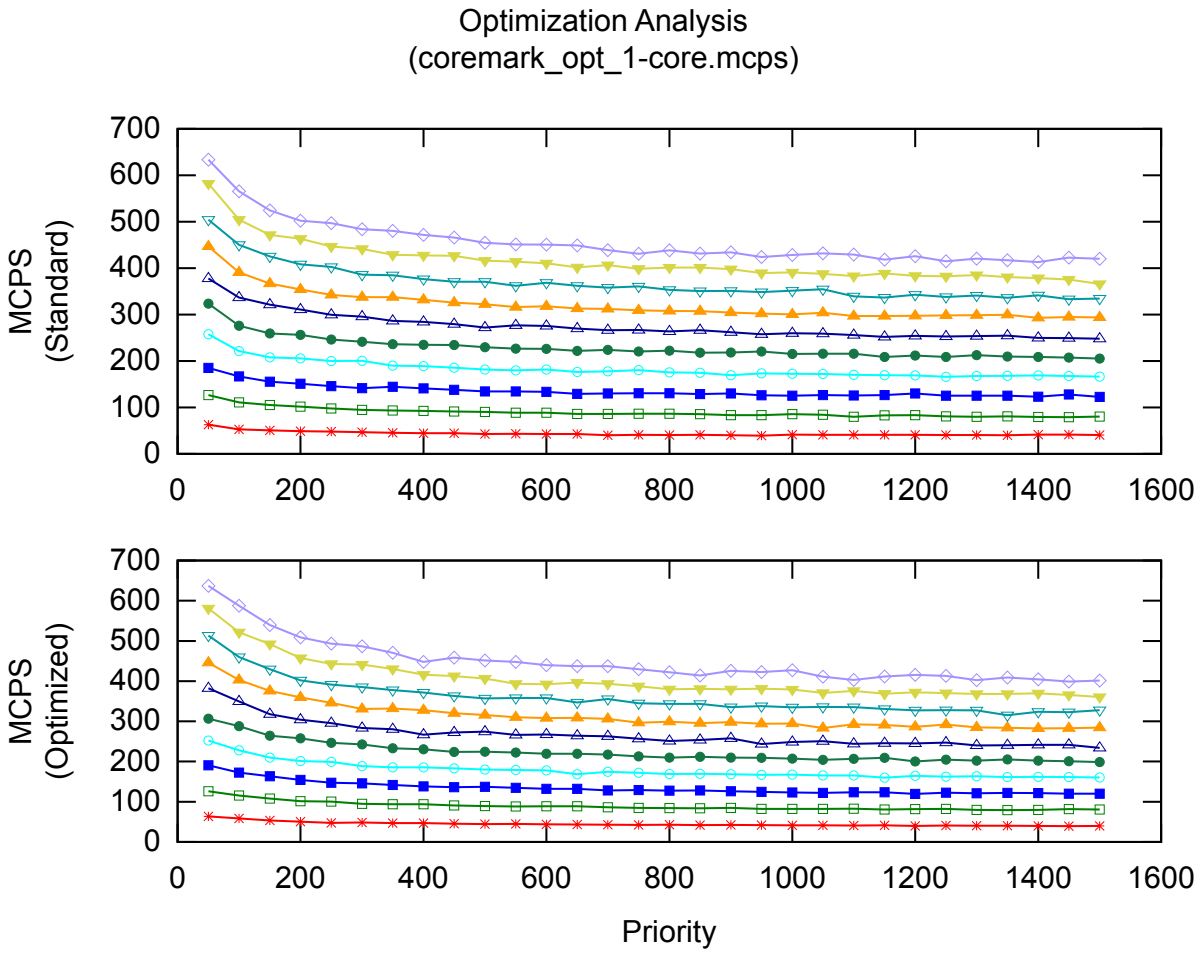


Figura 132: Comparativo de desempenho em MCPS do CoreMark

e o índice de erro da estimativa.

#### 6.5.1.2 Mergesort

A aplicação de Mergesort implementa um algoritmo clássico de ordenação que apresenta como principais características chave: o acesso sequencial aos dados e sua alta capacidade de paralelismo durante a sua execução. Nos experimentos realizados, o objetivo principal é verificar o impacto das otimizações nas operações de entrada e saída intensiva dos dispositivo de memória compartilhada da plataforma.

Na figura 134 é exibido os dados de estimativa de tempo simulado coletados das simulações com compilação padrão (parte superior) e com otimização de código (parte inferior). Pode ser visto que as estimativas de tempo simulado a partir da execução de código otimizado geraram valores com escala menor em comparação aos obtidos sem otimização, uma vez que a oti-

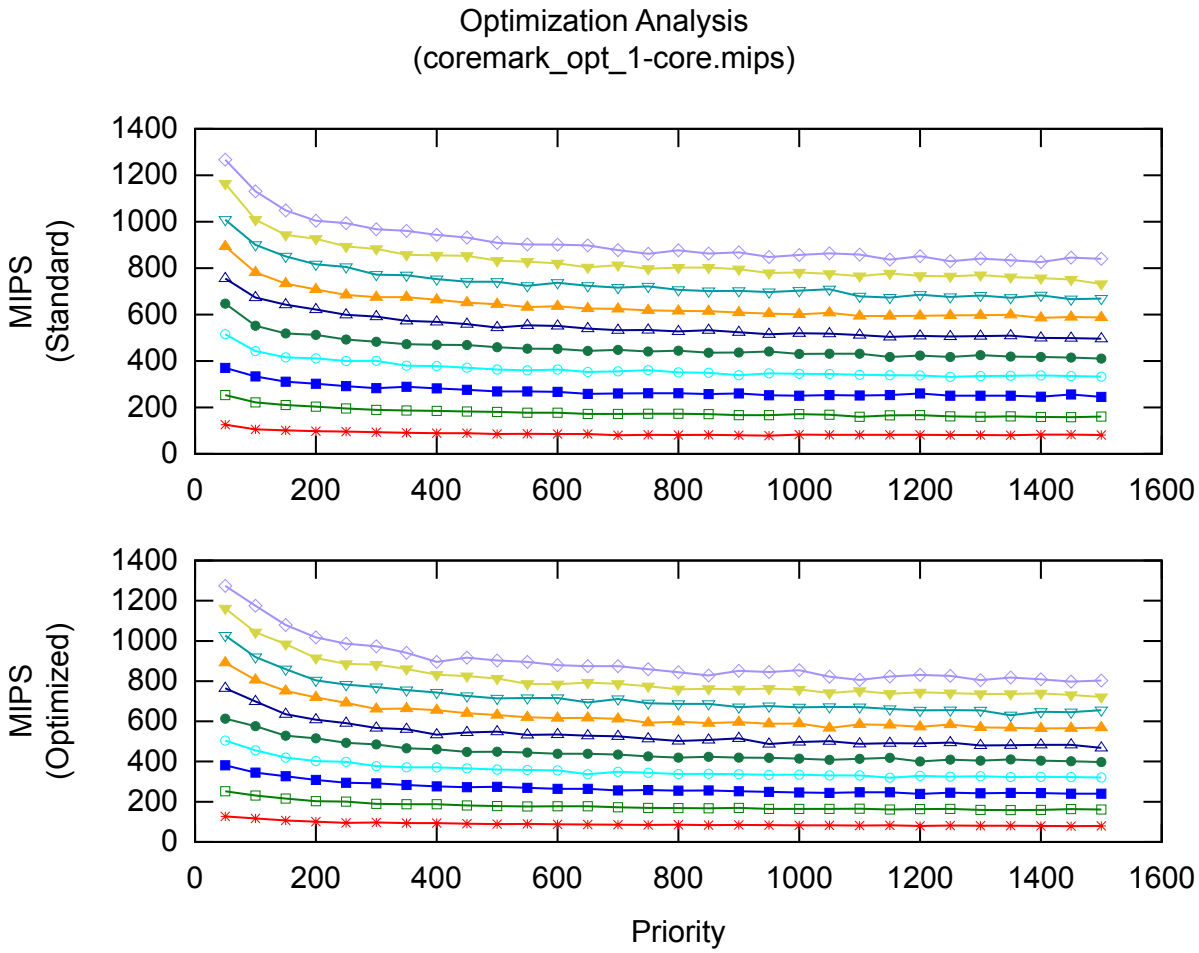


Figura 133: Comparativo de desempenho em MIPS do CoreMark

mização tem como objetivo reduzir o tempo de execução nativo de cada bloco básico da aplicação e isto implica em estimativas de tempo proporcionalmente menores.

A comparação dos desempenhos em MCPS e MIPS podem ser visualizados nas figuras 135 e 136, respectivamente, onde em cada figura a parte superior representa os dados sem otimização e a parte inferior os dados com utilização de otimização. Não é possível perceber facilmente as diferenças de desempenho, devido a escala utilizada, mas esta análise será feita utilizando os parâmetros de ajuste e de prioridade que foram calibrados com valores 23 e 2.038 na simulação sem otimização, respectivamente.

Para fins de comparação, é mantido o valor do parâmetro de prioridade para realização de uma calibração para plataforma com otimização de código, gerando um parâmetro de ajuste com valor 28 e atingindo um desempenho de cerca de 17 MCPS e 34 MIPS. Na simulação sem otimização de código

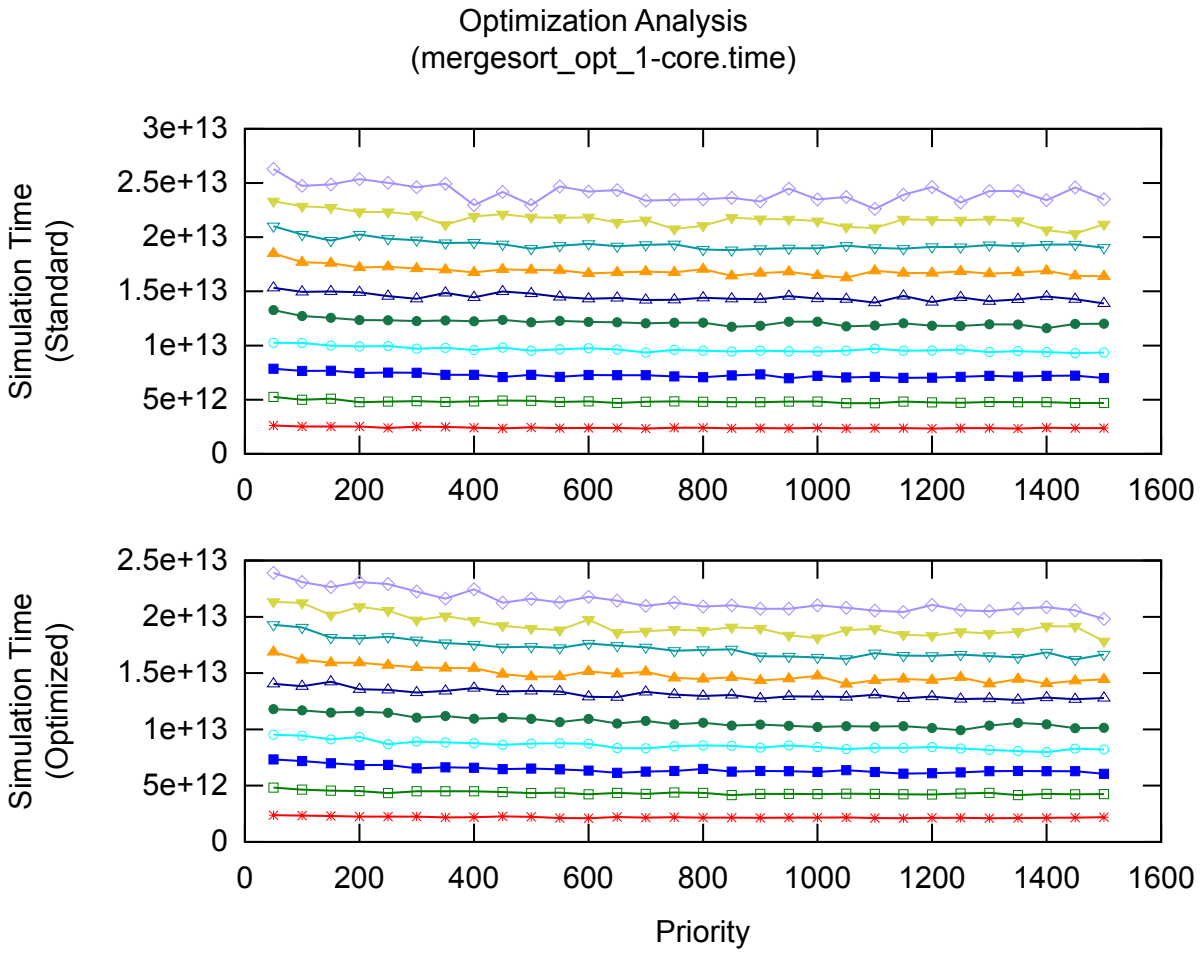


Figura 134: Comparativo de estimativa de tempo simulado do Mergesort

foram obtidos desempenhos de cerca de 14 MCPS e 28 MIPS, indicando que otimização de código pelo compilador melhorou o desempenho em aproximadamente 21%, sem alterar o comportamento das funções ou o nível de erro das estimativas geradas.

### 6.5.2 Execução Multiprocessada

A partir da análise dos resultados experimentais coletados são construídas curvas de desempenho com padrão tracejado de linha, através da aplicação da técnica de regressão não linear (79) que consiste em encontrar um ajuste de curva para os pontos fornecidos, minimizando o erro total da aproximação. Para encontrar uma curva que represente adequadamente o conjunto de pontos amostrados é preciso entender como o modelo de referência baseado em ISS calcula suas métricas de desempenho. É considerado que o tempo de execução da plataforma virtual é proporcional ao número de pro-

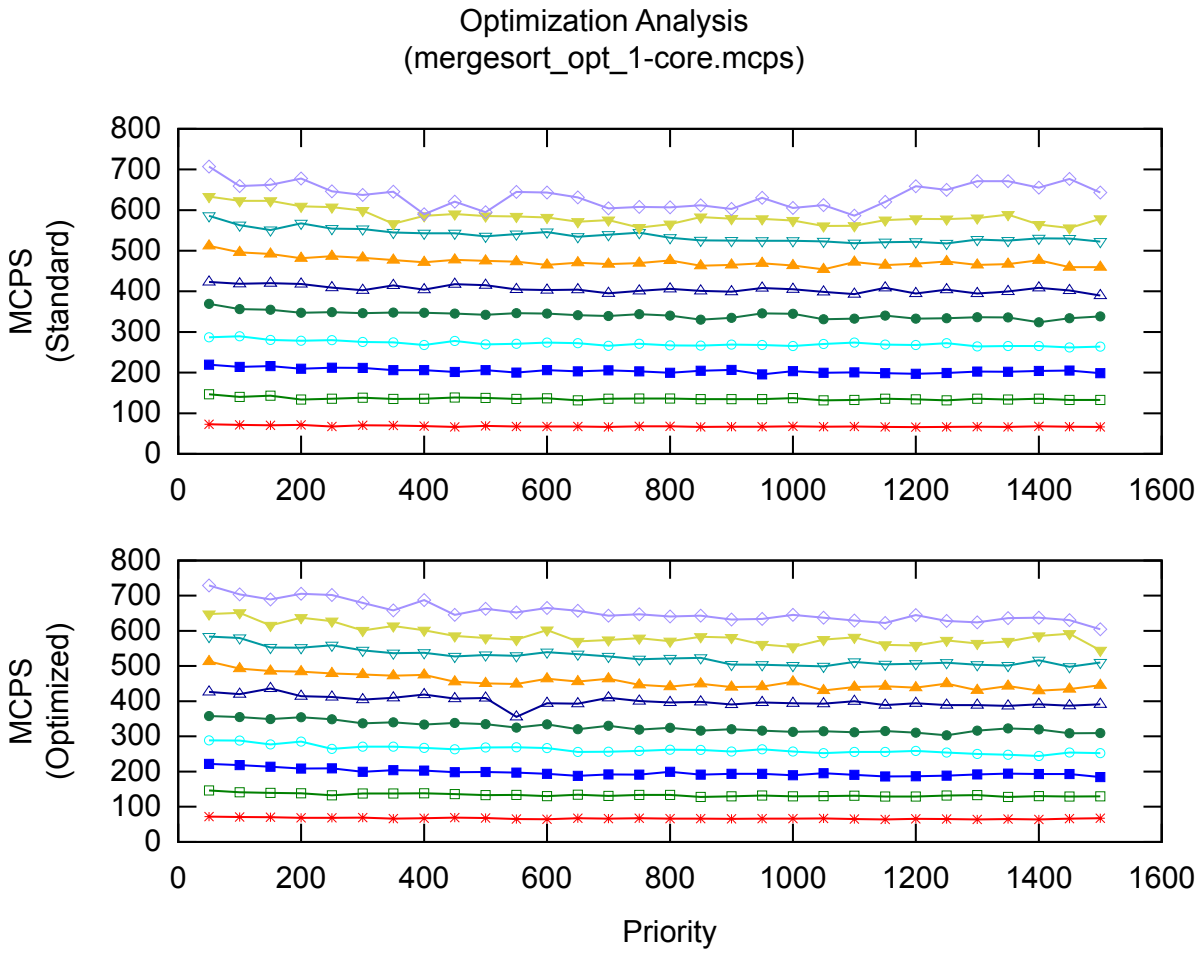


Figura 135: Comparativo de desempenho em MCPS do Mergesort

cessadores utilizados, desta forma, quanto mais instâncias sendo simuladas, maior é o tempo de execução do ambiente de simulação.

$$MCPS_{ISS}(N) = \frac{1}{N} \times \frac{Número\ de\ Ciclos}{Tempo\ de\ Execução} \times 10^{-6} = \frac{K_{Ciclos}}{N} \tag{6.10}$$

$$MIPS_{ISS}(N) = \frac{1}{N} \times \frac{Número\ de\ Instruções}{Tempo\ de\ Execução} \times 10^{-6} = \frac{K_{Instruções}}{N} \tag{6.11}$$

Nas fórmulas 6.10 e 6.11 é detalhado como uma plataforma com $N$ processadores baseados em ISS realiza o cálculo de desempenho médio para cada núcleo de processamento, adotando as métricas MCPS e MIPS, respectivamente. Para uma determinada aplicação, configuração de ciclo de relógio e taxa de execução de instruções, em ambas as métricas, a contagem do número de ciclos e de instruções podem ser representadas pelas constantes $K_{Ciclos}$ e $K_{Instruções}$.

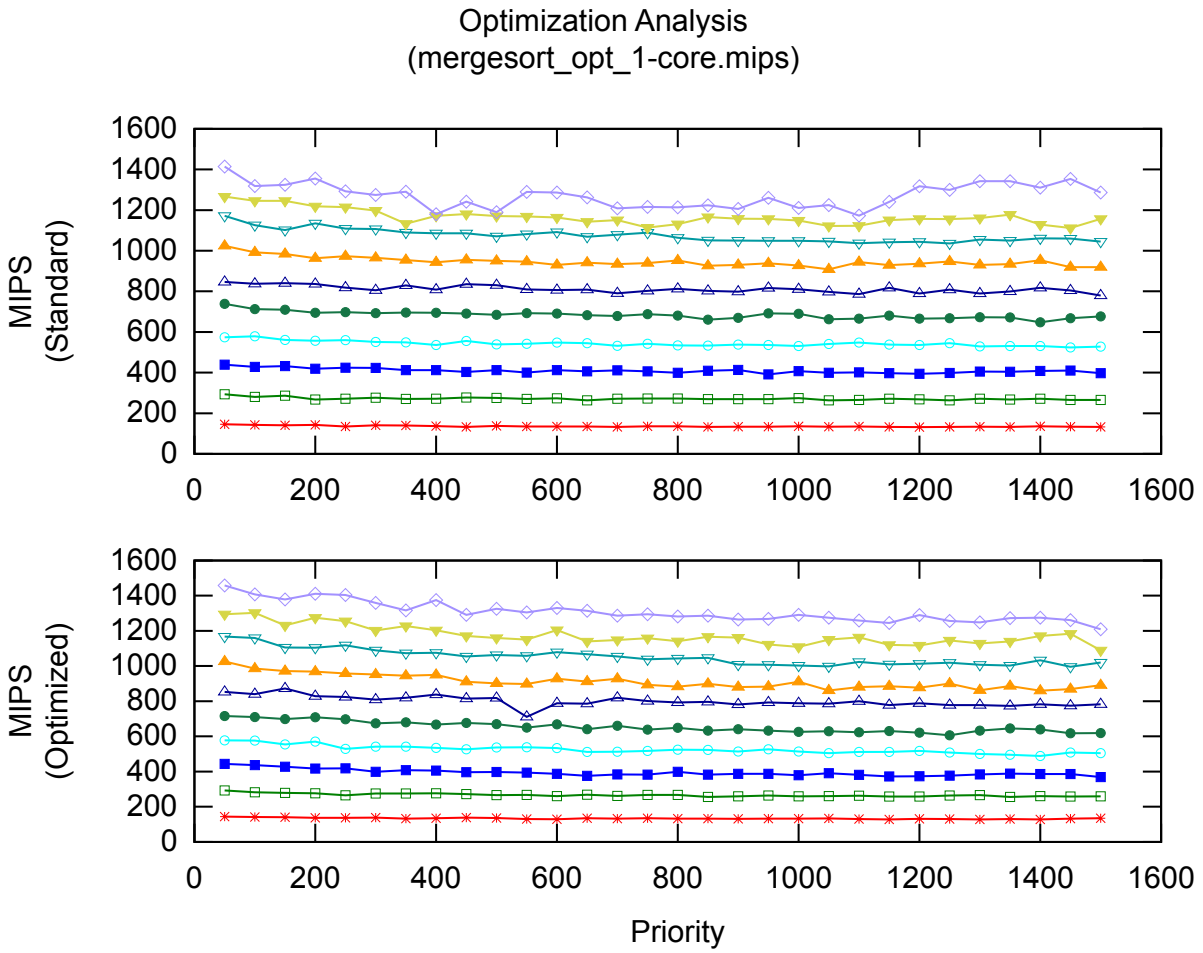


Figura 136: Comparativo de desempenho em MIPS do Mergesort

Após a realização destas reorganizações, as funções de desempenho em MCPS e MIPS correspondem a funções não lineares inversas cujo parâmetro é o número de processadores utilizados da plataforma. Por isto, em todas as regressões não lineares realizadas foi utilizada a função inversa $f(N) = \alpha/N + \beta$, para ambas as métricas de desempenho consideradas, onde $N$ é a quantidade de processadores utilizados na plataforma. Para encontrar os valores dos coeficientes $\alpha$ e $\beta$ das curvas de ajuste foi utilizada a ferramenta Gnuplot (80) para minimizar o erro de aproximação entre a curva obtida e os resultados de desempenho das simulações.

Em todos os gráficos referentes a modelagem proposta serão exibidos os valores médios obtidos para cada métrica de desempenho e seus respectivos desvios padrão associados, sendo representados graficamente por uma linha horizontal central e por linhas horizontais nas extremidades superior e inferior, respectivamente. Na realização dos experimentos são utilizadas plataformas com 1, 2, 4, 8 e 16 processadores, sendo geradas curvas de desempenho

com eixo referente ao número de processadores em escala logarítmica para facilitar a visualização dos resultados.

#### 6.5.2.1 CoreMark

Nesta subseção são apresentados os dados referentes a execução da aplicação CoreMark em plataformas multiprocessadas baseadas na abordagem proposta, com o objetivo de avaliar o comportamento dos sistemas em termo do desempenho em MCPS e MIPS, respectivamente, com relação as plataformas de referência que utilizam simuladores de instruções.

Tabela 26: Informações de desempenho do CoreMark (MCPS)

| # Núcleos | ISS | HdSC | Desvio Padrão |
|---|---|---|---|
| 1 | 1,07 MCPS | 360,37 MCPS | ± 4,00 MCPS |
| 2 | 0,53 MCPS | 255,38 MCPS | ± 5,77 MCPS |
| 4 | 0,25 MCPS | 171,85 MCPS | ± 3,56 MCPS |
| 8 | 0,13 MCPS | 79,24 MCPS | ± 2,61 MCPS |
| 16 | 0,06 MCPS | 14,55 MCPS | ± 0,80 MCPS |

Os dados de desempenho em MCPS do CoreMark, para plataformas com diferentes quantidades de núcleos de processamento, podem ser visualizados na tabela 26. Os valores referentes ao trabalho proposto (HdSC) representam um valor médio do desempenho por núcleo de processamento com seus respectivos desvios padrão e os desempenhos obtidos pela plataforma de referência (ISS).

Na figura 137 é exibida a curva contendo os desempenhos obtidos pela plataforma baseada em ISS (parte superior) e as estimativas de desempenho geradas pela modelagem proposta (parte inferior), utilizando a métrica de desempenho MCPS. Aplicando a metodologia definida, foi realizada uma regressão não linear a partir dos dados estatísticos coletados nas simulações para minimizar o erro de aproximação do conjunto de pontos com a curva prevista pelo modelo teórico.

O comportamento observado apresenta uma característica linear na escala considerada, diferente da previsão teórica de desempenho, e isto se deve basicamente pelo fato do ciclo de relógio da plataforma não ser considerado diretamente como um parâmetro da modelagem, sendo utilizado o

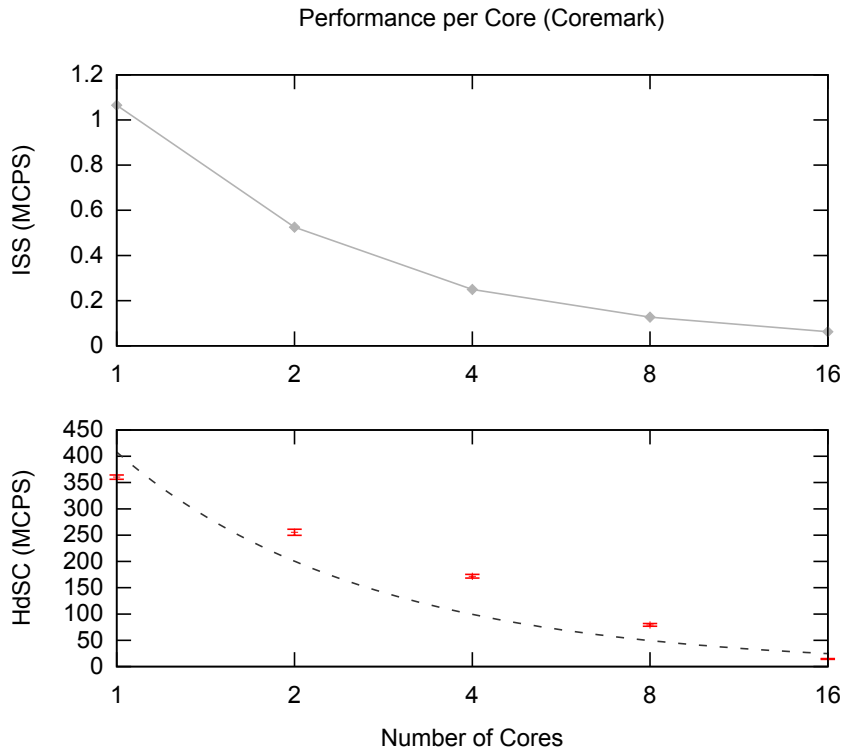


Figura 137: Gráfico de desempenho do CoreMark (MCPS)

tamanho do ciclo de instrução mais rápida da arquitetura.

Apesar desta diferença no comportamento da curva, as taxas de desempenho obtidas são consistentes com o modelo baseado em ISS, fornecendo uma tendência de redução de desempenho por núcleo a medida que a plataforma recebe mais processadores.

Tabela 27: Informações de desempenho do CoreMark (MIPS)

| # Núcleos | ISS | HdSC | Desvio Padrão |
|---|---|---|---|
| 1 | 0,65 MIPS | 720,73 MIPS | ± 8,00 MIPS |
| 2 | 0,32 MIPS | 350,16 MIPS | ± 7,91 MIPS |
| 4 | 0,15 MIPS | 161,64 MIPS | ± 3,35 MIPS |
| 8 | 0,08 MIPS | 68,70 MIPS | ± 2,26 MIPS |
| 16 | 0,04 MIPS | 25,50 MIPS | ± 1,41 MIPS |

Nos experimentos realizados com plataformas com múltiplos processadores e adotando a métrica MIPS podem ser vistos na tabela 27 os valores de desempenho médio obtidos pela abordagem proposta (HdSC) com seus valores de desvio padrão, assim como os resultados de desempenho gerados pela plataforma de referência (ISS).

Nas curvas de desempenho da figura 138 podem ser vistos, utilizando a mé-

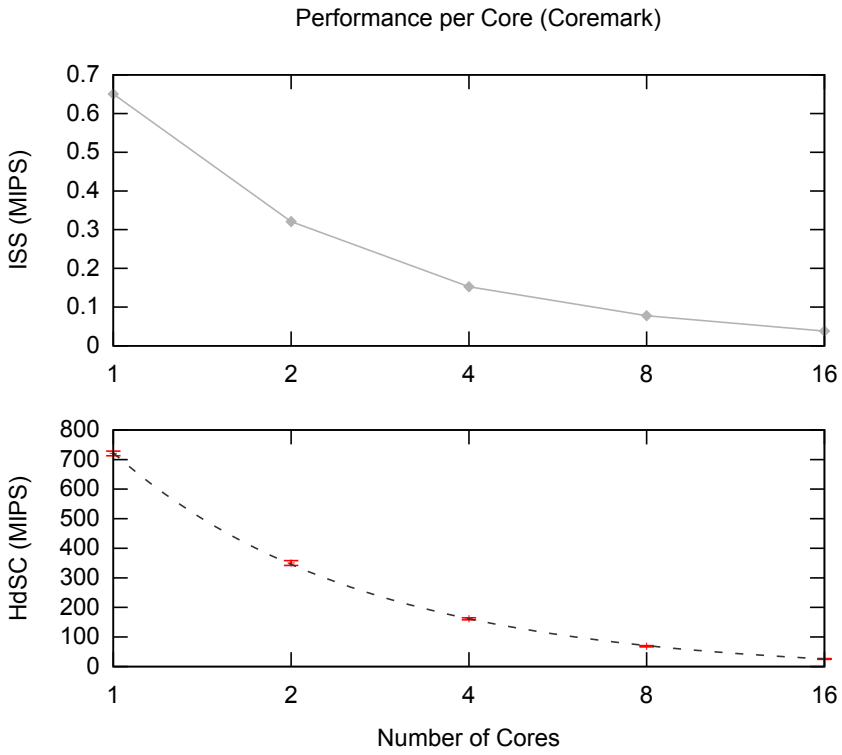


Figura 138: Gráfico de desempenho do CoreMark (MIPS)

trica MIPS, os desempenhos das plataformas baseadas em ISS (parte superior) e da modelagem proposta (parte inferior). Utilizando a metodologia definida, é realizada uma regressão não linear com os dados estatísticos coletados para ajustar os pontos experimentais com a curva teórica prevista.

Os resultados ilustram uma alta fidelidade ao comportamento observado no modelo ISS, decorrente do fato da modelagem proposta utilizar como parâmetro de menor atraso o tempo de execução da instrução de máquina.

#### 6.5.2.2 Mergesort

Os diversos experimentos realizados com plataformas multiprocessadas baseadas na abordagem proposta são comparados nesta subseção com os resultado obtidos das plataformas de referência que utilizam um modelo ISS. A análise comparativa é feita utilizando as métricas MCPS e MIPS, respectivamente, como forma de avaliar o comportamento dos sistemas.

Durante os experimentos realizados foram coletadas informações sobre o desempenho em MCPS da aplicação Mergesort, sendo listados valores médios de desempenho para abordagem proposta (HdSC), com seus respecti-

Tabela 28: Informações de desempenho do Mergesort (MCPS)

| # Núcleos | ISS | HdSC | Desvio Padrão |
|---|---|---|---|
| 1 | 2,05 MCPS | 14,10 MCPS | ± 1,05 MCPS |
| 2 | 2,30 MCPS | 14,71 MCPS | ± 0,79 MCPS |
| 4 | 2,77 MCPS | 18,13 MCPS | ± 1,06 MCPS |
| 8 | 3,32 MCPS | 21,95 MCPS | ± 1,02 MCPS |
| 16 | 3,92 MCPS | 23,38 MCPS | ± 1,12 MCPS |

vos desvios padrão, e os valores de desempenho obtidos nas plataformas de referência baseadas em ISS que podem ser vistos na tabela 28.

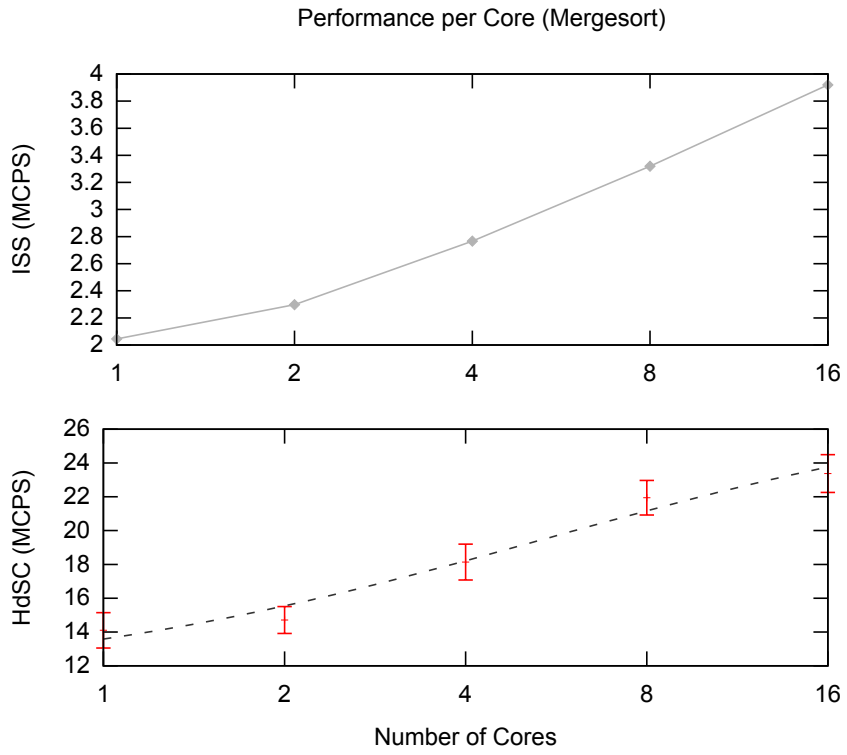


Figura 139: Gráfico de desempenho do Mergesort (MCPS)

Na figura 139 pode ser visto o desempenho de cada instância de processamento para as diferentes configurações de plataforma, considerando os modelos baseados em ISS (parte superior) e os modelos baseados na abordagem proposta (parte inferior), utilizando a métrica MCPS. Pode ser visto no gráfico as informações referentes aos dados estatísticos coletados e é feita a regressão não linear destes pontos sobre o modelo teórico de tempo previsto.

É observado que a curva obtida apresenta o mesmo comportamento obtido pelas plataformas baseadas em ISS, apesar da maior dispersão dos desvios padrão associados e da parametrização do modelo não considerar di-

retamente o tempo de ciclo de relógio nos experimentos. Este comportamento é resultado de uma aplicação de entrada e saída intensiva que consome grande parte do seu tempo simulado acessando a unidade de memória compartilhada, que não tem o tempo simulado estimado pelo núcleo de simulação proposto, mas sim pelos atrasos modelados pelos componentes da plataforma.

Tabela 29: Informações de desempenho do Mergesort (MIPS)

| # Núcleos | ISS | HdSC | Desvio Padrão |
|---|---|---|---|
| 1 | 0,65 MIPS | 28,20 MIPS | ± 2,10 MIPS |
| 2 | 0,39 MIPS | 21,51 MIPS | ± 1,16 MIPS |
| 4 | 0,21 MIPS | 20,87 MIPS | ± 1,22 MIPS |
| 8 | 0,12 MIPS | 21,08 MIPS | ± 0,98 MIPS |
| 16 | 0,06 MIPS | 20,30 MIPS | ± 0,97 MIPS |

Na tabela 29 são listados os valores de desempenho em MIPS da aplicação Mergesort para diferentes configurações de plataformas, com valores médios de desempenho para a abordagem proposta (HdSC) e seus respectivos desvios padrão, além dos desempenhos atingidos pelos processadores baseados em ISS.

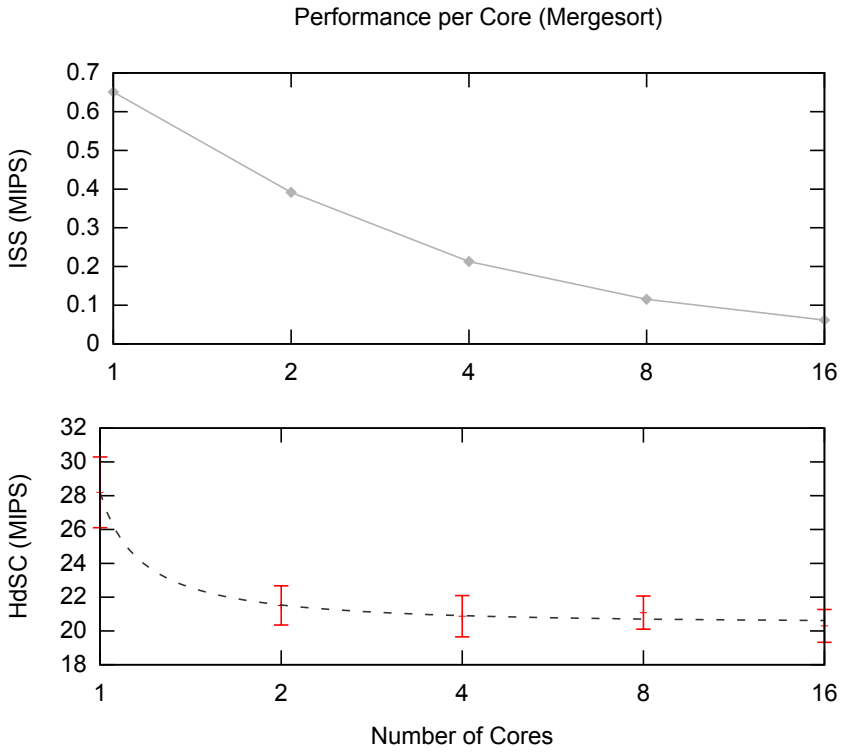


Figura 140: Gráfico de desempenho do Mergesort (MIPS)

Utilizando a métrica MIPS, podem ser observadas na figura 140 as curvas de desempenho para diferentes configurações de plataforma baseadas em ISS (parte superior) e baseadas na modelagem proposta (parte inferior). Aplicando a regressão não linear dos pontos experimentais sobre a curva teórica prevista é obtida uma curva de desempenho que atende aos limites de desvio padrão calculados pela análise estatística dos dados coletados.

É um fato que os desvios padrão associados são mais acentuadas do que os vistos nos experimentos anteriores, decorrentes em grande parte da aplicação em questão possuir uma característica de entrada e saída intensiva. Apesar disto, os resultados demonstram uma tendência de redução de desempenho em MIPS por núcleo de processamento, a medida que mais processadores são incluídos na plataforma, sendo consistentes com as simulações realizadas com plataformas baseadas em ISS.

# 7 Conclusão

Neste capítulo é feita uma sistematização dos principais tópicos discutidos ao longo do texto, retomando os conceitos chave abordados, os objetivos propostos e as contribuições geradas, exibindo as limitações consideradas e os potenciais trabalhos futuros para esta linha de pesquisa.

## 7.1 Contribuições e Objetivos

As primeiras páginas deste trabalho foram dedicadas a apresentar ao leitor as motivações, os objetivos propostos e que lacunas existem no estado da arte considerado. A ideia central das motivações está focada na melhoria de produtividade no projeto de sistemas que possuem software dependente do hardware, observando o cenário atual e projeções futuras, através de referências aos trabalhos relacionados. A partir destas informações, foram delineados um conjunto de objetivos para gerar contribuições relevantes nos cenários observados.

A definição destes objetivos levaram em consideração as lacunas existentes no estado da arte, ou seja, oportunidades para proposição de novas técnicas e mecanismos para problemas relevantes ainda não completamente resolvidos. Durante a concepção e proposição desta abordagem foram identificados requisitos para implementação das funcionalidades propostas, em grande parte decorrentes da execução nativa do software combinado com um ambiente de execução em plataforma virtual. A possibilidade de utilização de ferramentas nativas de desenvolvimento para a arquitetura alvo e a compatibilidade do código fonte do software para compilação cruzada agregaram bastante valor ao trabalho proposto, mas apresentaram importantes desafios ao longo de sua implementação.

A principal contribuição deste trabalho foi a proposição de uma modelagem, com alto nível de abstração de sistemas uniprocessados e multiprocessados, focada em execução nativa do software e suportando o desenvolvimento nativo de HdS através de uma integração com plataformas virtuais. Para suportar estes novos componentes criados, foi proposto um núcleo de simulação para sincronização, para interface com os dispositivos da plataforma e para realização de estimativas de tempo simulado. Uma importante contribuição da abordagem proposta, que a destaca dos trabalhos relacionados, é a realização de estimativa de tempo simulado a partir do tempo de execução nativa do software, sem necessidade de anotação de código com informações sobre uma determinada plataforma. As técnicas propostas estabelecem um modelo teórico parametrizável para estimativa de tempo simulado que permite avaliar o comportamento completo do sistema, demandando poucas simulações para sua calibração.

Para a avaliação e a validação de todos os conceitos propostos, foram aplicados um amplo conjunto de experimentos para demonstrar que todas as funcionalidades propostas foram implementadas e atendem diferentes áreas de aplicação. Foram utilizados métodos quantitativos para comparação dos resultados obtidos com os trabalhos relacionados, sempre enfatizando a melhoria de desempenho com média de cerca de 1.000 vezes e as baixas margens de erro médio em torno de aproximadamente 3% nas simulações realizadas, com relação ao uma plataforma de referência baseada em ISS.

## 7.2 Limitações

As principais limitações estão relacionadas com o uso de modelos de sistema bem comportados, decorrentes em grande parte do uso de plataformas virtuais de alto nível de abstração com comunicação em nível de transação (TLM) (26) e sem suporte para modelagem de memória cache. Mesmo utilizando um modelo de referência baseado em ISS, sua descrição funcional tem o foco na execução das instruções e não em aspectos como: estágios de pipeline, predição especulativa de desvios ou execução fora de ordem das instruções. Mais uma vez, pela proposta apresentada e pelo nível de detalhe empregado, estas limitações são mandatórias para que modelos possuam um

alto desempenho de simulação.

É importante enfatizar que o escopo deste trabalho está delimitado a interface de programação POSIX (60) que é utilizada pela maioria absoluta dos sistemas operacionais, como Linux, Mac OS, Solaris e Windows, por exemplo. É possível utilizar outras interfaces de programação disponíveis nativamente, bastando que a biblioteca necessária seja incluída e compilada em conjunto com o sistema. Todas as peculiaridades de cada SO que vão além das interfaces padronizadas, como o desenvolvimento de gerenciadores de dispositivos (drivers), são suportadas, mas precisam ser modeladas pelo projetista do sistema e podem ser reutilizadas em outros contextos de utilização.

## 7.3 Trabalhos Futuros

Após o detalhamento da abordagem proposta e de sua validação, é interessante que um conjunto de trabalhos futuros e oportunidades de melhoria sejam listados. Enumerar estes pontos é importante para auxiliar a definição de trabalhos que necessitem de plataformas rápidas de alto nível de abstração para serem bem sucedidos ou que desejem aprimorar as técnicas propostas. Dentre algumas destas possibilidades de novos trabalhos ou de aperfeiçoamentos, podem ser citadas:

- Conceber uma modelagem de alto nível para caches, permitindo uma avaliação dos diversos aspectos do projeto, como definição da capacidade, políticas de substituição dos blocos e medição das taxas de faltas para diferentes aplicações. Além destas funcionalidades, esta modelagem deve ser capaz de suportar mecanismos de coerência de cache que são essenciais em plataformas multiprocessadas;

- Automatização do processo de escolha do parâmetro de configuração de granularidade $P$ do sistema para simplificar a etapa de calibração do modelo e não demandar do projetista um conhecimento mais aprofundado da aplicação em questão e da própria infraestrutura de simulação. Isto evita que o projetista precise definir, baseado em sua experiência, qual o valor do parâmetro de granularidade mais adequado para que os eventos da plataforma sejam tratados adequadamente e que tam-

bém seja maximizado o desempenho da simulação;

- Aprimorar a interação e a própria linguagem SLDL para executar os modelos de sistema com maior grau de paralelismo. Na atual implementação da SLDL SystemC (31) existem gargalos decorrentes de seu sistema sequencial de escalonamento que impõe uma retenção durante a simulação dos múltiplos núcleos de processamento;

- Desenvolver uma série de ferramentas de suporte para automatizar a incorporação do código fonte da aplicação, permitindo uma compilação do sistema para executar nos modelos construídos. Este passo, que no estágio de desenvolvimento atual é feito manualmente, traria aumento de produtividade e reduziria a chance de erros no processo de desenvolvimento;

- Permitir geração automática de modelos a partir de especificações de mais alto nível, como UML (81, 82), por exemplo. Assim é esperado que exista uma maior interoperabilidade com outras ferramentas de modelagem em nível de sistema e, consequentemente, maior adoção dos mecanismos e técnicas propostas.

Evidentemente esta lista não é exaustiva e existem outros tópicos que podem ser considerados como trabalhos futuros, mas estes foram os pontos que se mostraram mais pertinentes e possuem maior potencial para se beneficiar da abordagem proposta. Em todo o processo de desenvolvimento deste trabalho houve uma preocupação em deixar muito bem documentado todos os artefatos gerados, como forma de permitir que outras pessoas possam se beneficiar e continuar esta linha de pesquisa.

# Referências

(1) ECKER, W.; HEINEN, S.; VELTEN, M. Using a dataflow abstracted virtual prototype for hds-design. In: *Proceedings of the 2009 Asia and South Pacific Design Automation Conference*. Piscataway, NJ, USA: IEEE Press, 2009. (ASP-DAC '09), p. 293–300. ISBN 978-1-4244-2748-2. Disponível em: <http://portal.acm.org/citation.cfm?id=1509633.1509712>.

(2) WANG, Z.; HERKERSDORF, A. Software performance simulation strategies for high-level embedded system design. *Perform. Eval.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 67, p. 717–739, August 2010. ISSN 0166-5316. Disponível em: <http://dx.doi.org/10.1016/j.peva.2009.07.003>.

(3) LO, C.-K. et al. Cycle-count-accurate processor modeling for fast and accurate system-level simulation. In: *Proceedings of the Conference on Design, Automation and Test in Europe*. (s.n.), 2011. (DATE '11). Disponível em: <http://logos.cs.nthu.edu.tw/blog/paper/1103_DATE_Lo.PDF>.

(4) RAZAGHI, P.; GERSTLAUER, A. Automatic timing granularity adjustment for host-compiled software simulation. In: *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*. (S.l.: s.n.), 2012. p. 567 –572. ISSN 2153-6961.

(5) GERSTLAUER, A. et al. Abstract system-level models for early performance and power exploration. In: *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*. (S.l.: s.n.), 2012. p. 213 –218. ISSN 2153-6961.

(6) YEH, T.-C.; CHIANG, M.-C. On the interfacing between qemu and systemc for virtual platform construction: Using dma as a case. *J. Syst. Archit.*, Elsevier North-Holland, Inc., New York, NY, USA, v. 58, n. 3-4, p. 99–111, mar. 2012. ISSN 1383-7621. Disponível em: <http://dx.doi.org/10.1016/j.sysarc.2012.02.002>.

(7) KRAEMER, S. et al. Hysim: a fast simulation framework for embedded software development. In: *Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*. New York, NY, USA: ACM, 2007. (CODES+ISSS '07), p. 75–80. ISBN 978-1-59593-824-4. Disponível em: <http://doi.acm.org/10.1145/1289816.1289837>.

(8) KRAUSE, M. et al. Combination of instruction set simulation and abstract rtos model execution for fast and accurate target software evaluation. In: *Proceedings of the 6th IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis*. New York, NY, USA: ACM, 2008. (CODES+ISSS '08), p. 143–148. ISBN 978-1-60558-470-6. Disponível em: <http://doi.acm.org/10.1145/1450135.1450168>.

(9) LISBOA, E. et al. A design flow based on a domain specific language to concurrent development of device drivers and device controller simulation models. In: *Proceedings of th 12th International Workshop on Software and Compilers for Embedded Systems*. New York, NY, USA: ACM, 2009. (SCOPES '09), p. 53–60. ISBN 978-1-60558-696-0. Disponível em: <http://portal.acm.org/citation.cfm?id=1543820.1543830>.

(10) SCHIRNER, G.; GERSTLAUER, A.; DöMER, R. Automatic generation of hardware dependent software for mpsocs from abstract system specifications. In: *Proceedings of the 2008 Asia and South Pacific Design Automation Conference*. Los Alamitos, CA, USA: IEEE Computer Society Press, 2008. (ASP-DAC '08), p. 271–276. ISBN 978-1-4244-1922-7. Disponível em: <http://portal.acm.org/citation.cfm?id=1356802.1356870>.

(11) SCHIRNER, G.; GERSTLAUER, A.; DöMER, R. Fast and accurate processor models for efficient mpsoc design. *ACM Trans. Des. Autom. Electron. Syst.*, ACM, New York, NY, USA, v. 15, p. 10:1–10:26, March 2010. ISSN 1084-4309. Disponível em: <http://doi.acm.org/10.1145/1698759.1698760>.

(12) ITRS. *The International Technology Roadmap for Semiconductors - System Drivers*. 2011. Disponível em: <http://www.itrs.net>.

(13) ITRS. *The International Technology Roadmap for Semiconductors - Design*. 2011. Disponível em: <http://www.itrs.net>.

(14) GARTNER. *Gartner Says More than 1 Billion PCs In Use Worldwide and Headed to 2 Billion Units by 2014*. 2008. Disponível em: <http://www.gartner.com/newsroom/id/703807>.

(15) GARTNER. *Gartner Says Western Europe PC Market Declined 19 Percent in Second Quarter of 2011*. 2011. Disponível em: <http://www.gartner.com/newsroom/id/1769215>.

(16) IHS. *IHS: 2011 embedded wireless market to grow 35%*. 2011. Disponível em: <http://www.eetimes.com/document.asp?doc_id=1259272>.

(17) SANGIOVANNI-VINCENTELLI, A.; MARTIN, G. Platform-based design and software design methodology for embedded systems. *IEEE Des. Test*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 18, n. 6, p. 23–33, nov. 2001. ISSN 0740-7475. Disponível em: <http://dx.doi.org/10.1109/54.970421>.

(18) DöMER, R.; GERSTLAUER, A.; MüLLER, W. Introduction to hardware-dependent software design hardware-dependent software for multi- and many-core embedded systems. In: *Proceedings of the 2009 Asia and South Pacific Design Automation Conference*. Piscataway, NJ, USA: IEEE Press, 2009. (ASP-DAC '09), p. 290–292. ISBN 978-1-4244-2748-2. Disponível em: <http://portal.acm.org/citation.cfm?id=1509633.1509711>.

(19) YAGI, H. et al. The wild west: conquest of complex hardware-dependent software design. In: *Proceedings of the 46th Annual Design Automation Conference*. New York, NY, USA: ACM, 2009. (DAC '09), p. 878–879. ISBN 978-1-60558-497-3. Disponível em: <http://doi.acm.org/10.1145/1629911.1630135>.

(20) ECKER, W.; DöMER, R.; MüLLER, W. *Hardware-dependent Software: Principles and Practice*. Dordrecht, GX, Netherlands: Springer Science + Business Media B.V., 2009. ISBN 9781402094354.

(21) KADAV, A.; SWIFT, M. M. Understanding modern device drivers. In: *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*. New York, NY, USA: ACM, 2012. (ASPLOS '12), p. 87–98. ISBN 978-1-4503-0759-8. Disponível em: <http://doi.acm.org/10.1145/2150976.2150987>.

(22) BOKHARI, S. The linux operating system. *Computer*, v. 28, n. 8, p. 74–79, 1995. ISSN 0018-9162.

(23) IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tools Flows. *IEEE Std 1685-2009*, p. C1–360, 2010.

(24) MOORE, G. E. Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff. *Solid-State Circuits Society Newsletter, IEEE*, v. 11, n. 5, p. 33–35, 2006. ISSN 1098-4232.

(25) YU, H.; DöMER, R.; GAJSKI, D. Embedded software generation from system level design languages. In: *Proceedings of the 2004 Asia and South Pacific Design Automation Conference*. Piscataway, NJ, USA: IEEE Press, 2004. (ASP-DAC '04), p. 463–468. ISBN 0-7803-8175-0. Disponível em: <http://portal.acm.org/citation.cfm?id=1015090.1015207>.

(26) CAI, L.; GAJSKI, D. Transaction level modeling: an overview. In: *Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. New York, NY, USA: ACM, 2003. (CODES+ISSS '03), p. 19–24. ISBN 1-58113-742-7. Disponível em: <http://doi.acm.org/10.1145/944645.944651>.

(27) STALLINGS, W. *Computer Organization and Architecture: Designing for Performance*. Upper Saddle River, NJ, USA: Pearson Education, Inc., 2010. ISBN 9780136073734.

(28) KAMATH, R.; ALEXANDER, P.; BARTON, D. Sldl: a systems level design language. In: *ASIC/SOC Conference, 1999. Proceedings. Twelfth Annual IEEE International*. (S.l.: s.n.), 1999. p. 71–75.

(29) GAJSKI, D. D. et al. *System Design: A Practical Guide with Specc*. Norwell, MA, USA: Kluwer Academic Publishers, 2001. ISBN 0792373871.

(30) FUJITA, M.; NAKAMURA, H. The standard specc language. In: *System Synthesis, 2001. Proceedings. The 14th International Symposium on*. (S.l.: s.n.), 2001. p. 81–86.

(31) GROTKER, T. *System Design with SystemC*. Norwell, MA, USA: Kluwer Academic Publishers, 2002. ISBN 1402070721.

(32) IEEE Standard System C Language Reference Manual. *IEEE Std 1666-2005*, p. 1–423, 2006.

(33) BERGERON, J. *Writing Testbenches: Functional Verification of HDL Models*. (S.l.): Kluwer Academic Publishers, 2003. ISBN 9781402074011.

(34) SYSTEMS, C. D. *Catapult: Product Family Overview*. 2013. Disponível em: <http://calypto.com/en/products/catapult/overview>.

(35) SYSTEMS, F. D. *Cynthesizer 5*. 2013. Disponível em: <http://www.forteds.com/products/cynthesizer.asp>.

(36) VAHID, F. *Digital Design with RTL Design, VHDL, and Verilog*. Hoboken, NJ, USA: John Wiley and Sons, Inc., 2010. ISBN 9780470531082.

(37) IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, p. 1–2793, 2012.

(38) CORBET, J.; RUBINI, A.; KROAH-HARTMAN, G. *Linux Device Drivers*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2005. ISBN 0596005903.

(39) BAILEY, B.; MARTIN, G.; PIZIALI, A. *ESL Design and Verification: A Prescription for Electronic System-Level Methodology*. San Francisco, CA, USA: Elsevier, Inc., 2007. ISBN 9780123735515.

(40) MARTIN, G. Processor stew (review of processor description languages by p. mishra and n. dutt, eds.; 2008) (book reviews). *Design Test of Computers, IEEE*, v. 26, n. 2, p. 76–77, 2009. ISSN 0740-7475.

(41) HENNESSY, J. L.; PATTERSON, D. A. *Computer Architecture: A Quantitative Approach*. San Francisco, CA, USA: Elsevier, Inc., 2007. ISBN 9780123704900.

(42) HERLIHY, M.; SHAVIT, N. *The Art of Multiprocessor Programming*. Burlington, MA, USA: Elsevier, Inc., 2008. ISBN 9780123705914.

(43) AMDAHL, G. M. Validity of the single processor approach to achieving large scale computing capabilities, reprinted from the afips conference proceedings, vol. 30 (atlantic city, n.j., apr. 18 2013), afips press, reston, va., 1967, pp. 483;485, when dr. amdahl was at international business machines corporation, sunnyvale, california. *Solid-State Circuits Society Newsletter, IEEE*, v. 12, n. 3, p. 19–20, 2007. ISSN 1098-4232.

(44) JERRAYA, A. A.; WOLF, W. *Multiprocessor System-on-Chips*. San Francisco, CA, USA: Elsevier Inc., 2005. ISBN 012385251X.

(45) FLYNN, M. Very high-speed computing systems. *Proceedings of the IEEE*, v. 54, n. 12, p. 1901 – 1909, dec. 1966. ISSN 0018-9219.

(46) GAISLER, A. *Leon3 Processor*. 2013. Disponível em: <http://www.gaisler.com/index.php/products/processors/leon3>.

(47) JANG, M.; KIM, K.; KIM, K. The performance analysis of arm neon technology for mobile platforms. In: *Proceedings of the 2011 ACM Symposium on Research in Applied Computation*. New York, NY, USA: ACM, 2011. (RACS '11), p. 104–106. ISBN 978-1-4503-1087-1. Disponível em: <http://doi.acm.org/10.1145/2103380.2103401>.

(48) SPECTOR, A.; GIFFORD, D. The space shuttle primary computer system. *Commun. ACM*, ACM, New York, NY, USA, v. 27, n. 9, p. 872–900, set. 1984. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/358234.358246>.

(49) HALAAS, A. et al. A recursive misd architecture for pattern matching. *IEEE Trans. Very Large Scale Integr. Syst.*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 12, n. 7, p. 727–734, jul. 2004. ISSN 1063-8210. Disponível em: <http://dx.doi.org/10.1109/TVLSI.2004.830918>.

(50) DEVICES, A. M. *Global Provider of Innovative Graphics, Processors and Media Solutions*. 2013. Disponível em: <http://www.amd.com/>.

(51) CORPORATION, I. *Ultrabook, SmartPhone, Laptop, Desktop, Server, & Embedded-Intel*. 2013. Disponível em: <http://www.intel.com/>.

(52) HENNESSY, J. L.; PATTERSON, D. A. *Computer Architecture: A Quantitative Approach*. San Francisco, CA, USA: Elsevier, Inc., 2011. ISBN 9780123838735.

(53) KUMAR, R. et al. Heterogeneous chip multiprocessors. *Computer*, v. 38, n. 11, p. 32–38, 2005. ISSN 0018-9162.

(54) EXYNOS, S. *Samsung Primes Exynos 5 Octa for ARM big.LITTLE Technology with Heterogeneous Multi-Processing Capability*. 2013. Disponível em: <http://www.samsung.com/global/business/semiconductor/minisite/Exynos/news_Samsung_Primes_Exynos5Octa_for_ARM_bigLITTLE_Technology_with_Heterogeneous_Multi_Processing_Capability.html?ism=SASep0913Twitter2>.

(55) OBAIDAT, M. S.; BOUDRIGA, N. A. *Fundamentals of Performance Evaluation of Computer and Telecommunication Systems*. Hoboken, NJ, USA: John Wiley and Sons, Inc., 2010. ISBN 9780471269830.

(56) BELLARD, F. Qemu, a fast and portable dynamic translator. In: *Proceedings of the Annual Conference on USENIX Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2005. (ATEC '05), p. 41–41. Disponível em: <http://dl.acm.org/citation.cfm?id=1247360.1247401>.

(57) PRADO, B. et al. Hdsc: a fast and preemptive modeling for on host hds development. In: *Proceedings of the 24th symposium on Integrated circuits and systems design*. New York, NY, USA: ACM, 2011. (SBCCI '11), p. 179–184. ISBN 978-1-4503-0828-1. Disponível em: <http://doi.acm.org/10.1145/2020876.2020917>.

(58) PRADO, B. et al. Hdsc: a fast and preemptive modeling for on host hds development. *Journal of Integrated Circuits*

*and Systems*, v. 7, n. 1, p. 61–71, March 2012. Disponível em: <http://www.sbmicro.org.br/jics/html/artigos/vol7no1/6.pdf>.

(59) ANSI. *American National Standards Institute*. 2012. Disponível em: <http://www.ansi.org/>.

(60) IEEE. *IEEE SA - POSIX - Austin Joint Working Group*. 2012. Disponível em: <http://standards.ieee.org/develop/wg/POSIX.html>.

(61) MCCRACKEN, D. D.; REILLY, E. D. Backus-naur form (bnf). In: *Encyclopedia of Computer Science*. Chichester, UK: John Wiley and Sons Ltd. p. 129–131. ISBN 0-470-86412-5. Disponível em: <http://dl.acm.org/citation.cfm?id=1074100.1074155>.

(62) IEEE. *IEEE SA - POSIX*. 2013. Disponível em: <http://standards.ieee.org/develop/wg/POSIX.html>.

(63) CORMEN, T. H. et al. *Introduction to Algorithms*. Cambridge, MA, USA: The MIT Press, 2009. ISBN 9780262033848.

(64) SPARC. *Welcome to SPARC International, Inc.!* 2012. Disponível em: <http://www.sparc.org>.

(65) HASSANZADEH, I.; DARABI, M.; HASSANZADEH, S. Rapid prototyping of a manipulator mechanism using hardware in the loop (hil) simulators and comparing the results. In: *Proceedings of the 5th WSEAS international conference on Applications of electrical engineering*. Stevens Point, Wisconsin, USA: World Scientific and Engineering Academy and Society (WSEAS), 2006. (AEE'06), p. 191–195. ISBN 960-8457-42-4. Disponível em: <http://dl.acm.org/citation.cfm?id=1973925.1973967>.

(66) SCHLAGER, M.; OBERMAISSER, R.; ELMENREICH, W. A framework for hardware-in-the-loop testing of an integrated architecture. In: *Proceedings of the 5th IFIP WG 10.2 international conference on Software technologies for embedded and ubiquitous systems*. Berlin, Heidelberg: Springer-Verlag, 2007. (SEUS'07), p. 159–170. ISBN 3-540-75663-9, 978-3-540-75663-7. Disponível em: <http://dl.acm.org/citation.cfm?id=1778978.1778997>.

(67) CHANG, T. Dynamic characteristics rebuilding of hil simulation system and its validation. In: *Proceedings of the 2010 International Conference on Digital Manufacturing & Automation - Volume 01*. Washington, DC, USA: IEEE Computer Society, 2010. (ICDMA '10), p. 581–584. ISBN 978-0-7695-4286-7. Disponível em: <http://dx.doi.org/10.1109/ICDMA.2010.109>.

(68) ESTEC. *European Space Research and Technology Centre*. 2012. Disponível em: <http://www.esa.int/SPECIALS/ESTEC/index.html>.

(69) ESA. *European Space Agency Portal*. 2012. Disponível em: <http://www.esa.int/esaCP/>.

(70) AEROFLEX. *Aeroflex Gaisler AB*. 2012. Disponível em: <http://www.gaisler.com/>.

(71) AZEVEDO, R. et al. The archc architecture description language and tools. *Int. J. Parallel Program.*, Kluwer Academic Publishers, Norwell, MA, USA, v. 33, p. 453–484, October 2005. ISSN 0885-7458. Disponível em: <http://portal.acm.org/citation.cfm?id=1146113.1146115>.

(72) M., S. R.; COMMUNITY, T. G. D. *Using the GNU Compiler Collection For GCC version 4.2.1*. Boston, MA, USA: GNU Press, 2005.

(73) WOODBURY, G. *An Introduction to Statistics*. Pacific Grove, CA, USA: Thomson Learning, 2002. ISBN 0534377556.

(74) EEMBC. *EEMBC – The Embedded Microprocessor Benchmark Consortium*. 2012. Disponível em: <http://www.eembc.org/>.

(75) COREMARK. *CoreMark and EEMBC Benchmark*. 2013. Disponível em: <http://www.coremark.org/benchmark/index.php?pg=benchmark>.

(76) INC., X. *Spartan-6 FPGA SP605*. 2013. Disponível em: <http://www.xilinx.com/products/boards-and-kits/EK-S6-SP605-G.htm>.

(77) WEICKER, R. P. Dhrystone: a synthetic systems programming benchmark. *Commun. ACM*, ACM, New York, NY, USA, v. 27, n. 10, p. 1013–1030, out. 1984. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/358274.358283>.

(78) PROJECT, T. N. *Hardware Documentation - Machines DEC - VAX hardware reference*. 2013. Disponível em: <http://www.netbsd.org/docs/Hardware/Machines/DEC/vax/vax700.html>.

(79) JAIN, R. *Art of Computer Systems Performance Analysis Techniques for Experimental Design Measurements Simulation and Modeling*. New York, NY, USA: John Wiley ans Sons, Inc., 1991. ISBN 0471503363.

(80) WILLIAMS, T.; KELLEY, C. *gnuplot homepage*. 2013. Disponível em: <http://gnuplot.sourceforge.net/>.

(81) CHEN, R. et al. Uml for real. In: LAVAGNO, L.; MARTIN, G.; SELIC, B. (Ed.). Norwell, MA, USA: Kluwer Academic Publishers, 2003. cap. UML and platform-based design, p. 107–126. ISBN 1-4020-7501-4. Disponível em: <http://dl.acm.org/citation.cfm?id=886344.886350>.

(82) MUELLER, W. et al. Uml for esl design: basic principles, tools, and applications. In: *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*. New York, NY, USA: ACM, 2006. (ICCAD '06), p. 73–80. ISBN 1-59593-389-1. Disponível em: <http://doi.acm.org/10.1145/1233501.1233518>.

(83) GUTHAUS, M. R. et al. Mibench: A free, commercially representative embedded benchmark suite. In: *Proceedings of the Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop*. Washington, DC, USA: IEEE Computer Society, 2001. (WWC '01), p. 3–14. ISBN 0-7803-7315-4. Disponível em: <http://dx.doi.org/10.1109/WWC.2001.15>.

(84) JPEG. *Joint Photographic Experts Group*. 2012. Disponível em: <http://www.jpeg.org/jpeg/index.html>.

(85) GNU. *Ispell: Spell-checking for Unix*. 2012. Disponível em: <http://www.gnu.org/software/ispell/ispell.html>.

(86) SCHNEIER, B. Fast software encryption. In: *Cambridge Security Workshop Proceedings*. Cambridge, MA, USA: Springer-Verlag, 1994. p. 191–194. Disponível em: <http://www.schneier.com/paper-blowfish-fse.html>.

(87) DAEMEN, J.; RIJMEN, V. *The Design of Rijndael*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2002. ISBN 3540425802.

(88) EASTLAKE 3RD, D.; JONES, P. *US Secure Hash Algorithm 1 (SHA1)*. United States: RFC Editor, 2001.

(89) CUMMISKEY, P.; JAYANT, N. S.; FLANAGAN, J. L. Adaptive quantization in differential pcm coding of speech. In: *The Bell System Technical Journal, v52*. American Telephone and Telegraph Company, 1973. p. 1105–1118. Disponível em: <http://www.alcatel-lucent.com/bstj/vol52-1973/articles/bstj52-7-1105.pdf>.

(90) PETERSON, W.; BROWN, D. Cyclic codes for error detection. *Proceedings of the IRE*, v. 49, n. 1, p. 228 –235, jan. 1961. ISSN 0096-8390.

(91) STONE, H. R66-50 an algorithm for the machine calculation of complex fourier series. *Electronic Computers, IEEE Transactions on*, EC-15, n. 4, p. 680 –681, aug. 1966. ISSN 0367-7508.

(92) ETSI. *European Telecommunications Standards Institute*. 2012. Disponível em: <http://www.etsi.org/>.

# APÊNDICE A – Gramática de HdSC

Este apêndice é destinado a especificar a gramática da linguagem HdSC, utilizando o formato BNF (61). Os símbolos não terminais estão descritos entre delimitadores < > e os terminais estão descritos pelos delimitadores ' '. É importante destacar que o escopo desta gramática está delimitada a linguagem HdSC que utiliza a infraestrutura oferecida pelas linguagens C, C++ e SystemC, desta forma todos os símbolos não terminais não especificados nesta gramática são referentes a estas linguagens.

Código Fonte A.1: Gramática no formato BNF da linguagem HdSC

```
1  <software_declaration>
2       ::=
3       <software_template> <software_module>
4
5  <processor_declaration>
6       ::=
7       <processor_template> <processor_module>
8
9  <software_template>
10      ::=
11      'HSC_SOFTWARE_TEMPLATE' '(' <address_type> ',' <data_type>
            ')'
12
13 <software_module>
14      ::=
15      'HSC_SOFTWARE' '(' <identifier> ')' '{' <
            software_constructor> <statements> '}'
16
17 <processor_template>
```

```
18      ::=
19      'HSC_PROCESSOR_TEMPLATE' '(' <address_type> ',' <data_type>
            ')'
20
21  <processor_module>
22      ::=
23      'HSC_PROCESSOR' '(' <identifier> ')' '{' <
            processor_constructor> <statements> '}'
24
25  <software_constructor>
26      ::=
27      'HSC_SOFTWARE_CTOR' '(' <identifier> ')' '{' <statements>
            '}'
28
29  <processor_constructor>
30      ::=
31      'HSC_PROCESSOR_CTOR' '(' <identifier> ')' '{' <statements>
            '}'
32
33  <statements>
34      ::=
35      <c_cplusplus_systemc_statements>
36      |
37      <hdsc_statements> ';'
38
39  <hdsc_statements>
40      ::=
41      <software_main>
42      |
43      <software_global>
44      |
45      <software_encapsulation>
46      |
47      <software_register>
48      |
```

```
49      <software_register_field>
50      |
51      <software_register_range>
52      |
53      <software_register_declaration>
54      |
55      <software_register_custom_declaration>
56      |
57      <software_pointer_declaration>
58
59  <software_main>
60      ::=
61      'HSC_SOFTWARE_MAIN' '(' <identifier> ')'
62
63  <software_global>
64      ::=
65      'HSC_SOFTWARE_GLOBAL' '(' <data_type> ',' <identifier> ')'
66
67  <software_encapsulation>
68      ::=
69      'HSC_SOFTWARE_ENCAPSULATION' '(' <data_type> ',' <
          identifier> ',' <variadic> ')'
70
71  <software_register>
72      ::=
73      'HSC_SOFTWARE_REGISTER' '(' <identifier> ')' '{' <
          software_register_field> <software_register_map> '}'
74
75  <software_register_field>
76      ::=
77      'HSC_SOFTWARE_REGISTER_FIELD' '(' <identifier> ')'
78
79  <software_register_map>
80      ::=
```

```
81      'HSC_SOFTWARE_REGISTER_MAP' '(' <software_register_range>
            ')'
82
83  <software_register_range>
84      ::=
85      'HSC_SOFTWARE_REGISTER_RANGE' '(' <identifier> ',' <
            number_type> ',' <number_type> ')'
86
87  <software_register_declaration>
88      ::=
89      'HSC_SOFTWARE_REGISTER_DECLARATION' '(' <identifier> ')'
90
91  <software_register_custom_declaration>
92      ::=
93      'HSC_SOFTWARE_REGISTER_CUSTOM_DECLARATION' '(' <identifier>
            ',' <identifier> ')'
94
95  <software_pointer_declaration>
96      ::=
97      'HSC_SOFTWARE_POINTER_DECLARATION' '(' <identifier> ')'
98
99  <address_type>
100     ::=
101     <c_cplusplus_systemc_address_type>
102
103 <data_type>
104     ::=
105     <c_cplusplus_systemc_data_type>
106
107 <identifier>
108     ::=
109     <c_cplusplus_systemc_identifier>
110
111 <variadic>
112     ::=
```

```
113         <c_cplusplus_systemc_variadic_parameters>
114
115  <number_type>
116         ::=
117         <c_cplusplus_systemc_numeric_type>
```

# *APÊNDICE B – Aplicações MiBench*

Neste apêndice, todas as aplicações utilizadas do pacote MiBench são descritas e tem seus resultados detalhados, permitindo ao leitor uma análise mais aprofundada do comportamento da plataforma que está sendo avaliada sob a perspectiva de diferentes áreas de aplicação.

## B.1 MiBench Automotive

Neste conjunto de aplicações do MiBench são encontradas aplicações relevantes em sistemas embarcados da indústria automotiva, de tal maneira que seja possível avaliar a plataforma neste contexto de aplicação e que suas características sejam observadas.

### B.1.1 Basicmath

Atendendo a necessidade de se ter uma aplicação de referência de desempenho representativa para certos nichos de atuação, o conjunto de referência MiBench (83) foi desenvolvido para o contexto de sistemas embarcados, focando em diversas áreas, como a automotiva. Uma destas aplicações automotivas é a Basicmath que realiza operações matemáticas, que normalmente não são suportadas por hardware dedicado disponível em processadores embarcados.

Nas figuras 141 e 142, são exibidos os desempenhos em MCPS e MIPS atingido pela aplicação Basicmath no modelo proposto, usando a entrada pequena do MiBench. Pode ser observar que dentro do intervalo de ajuste de curvas de 100 (C0) a 1.000 (C9), com passos de 100 unidades, e de priorização

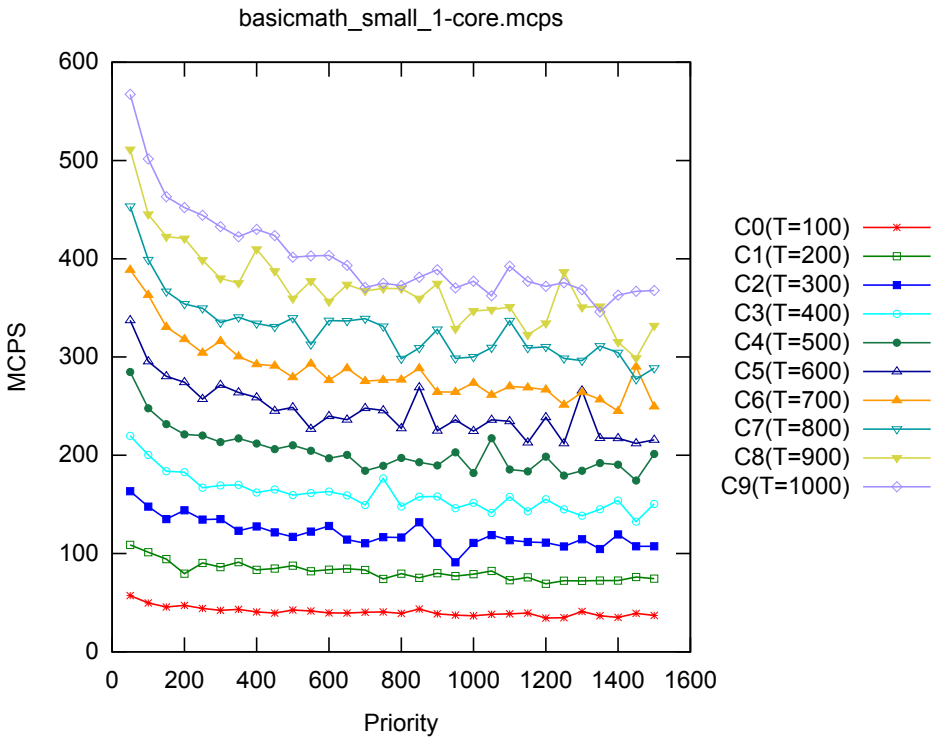


Figura 141: Desempenho em MCPS de Basicmath (Pequeno)

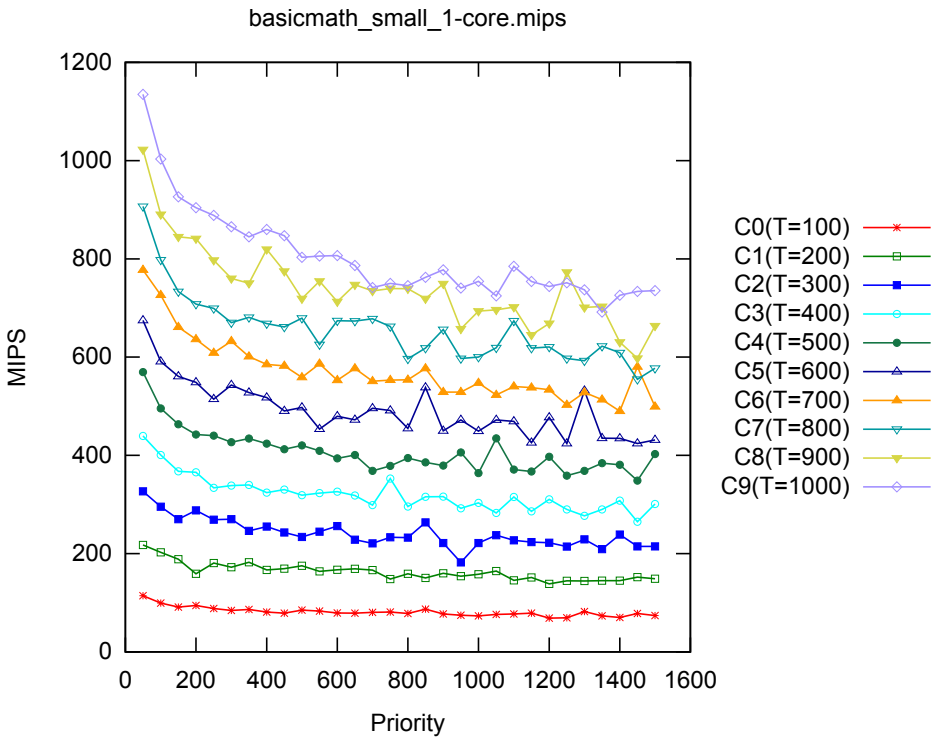


Figura 142: Desempenho em MIPS de Basicmath (Pequeno)

de 50 até 1.500, com passos de tamanho 50, foi atingido um pico de desempenho de cerca de 550 MCPS e 1.100 MIPS. Estes resultados representam uma melhoria de desempenho de 604 e 1.774 vezes, respectivamente, sobre o mo-

delo ISS que possui 0,91 MCPS e 0,62 MIPS de desempenho.

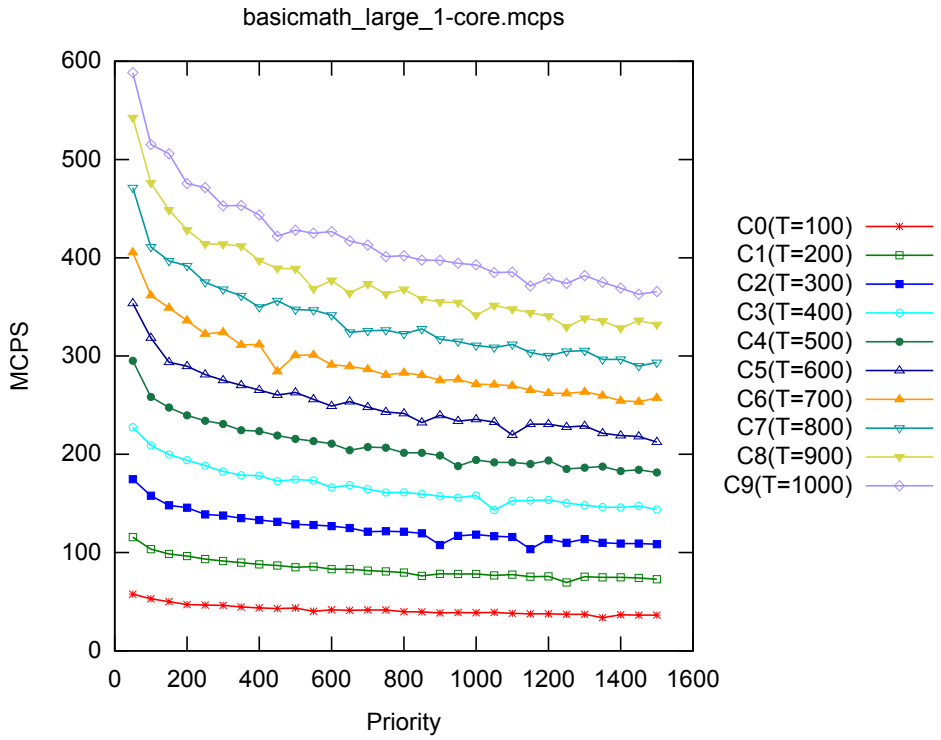


Figura 143: Desempenho em MCPS de Basicmath (Grande)

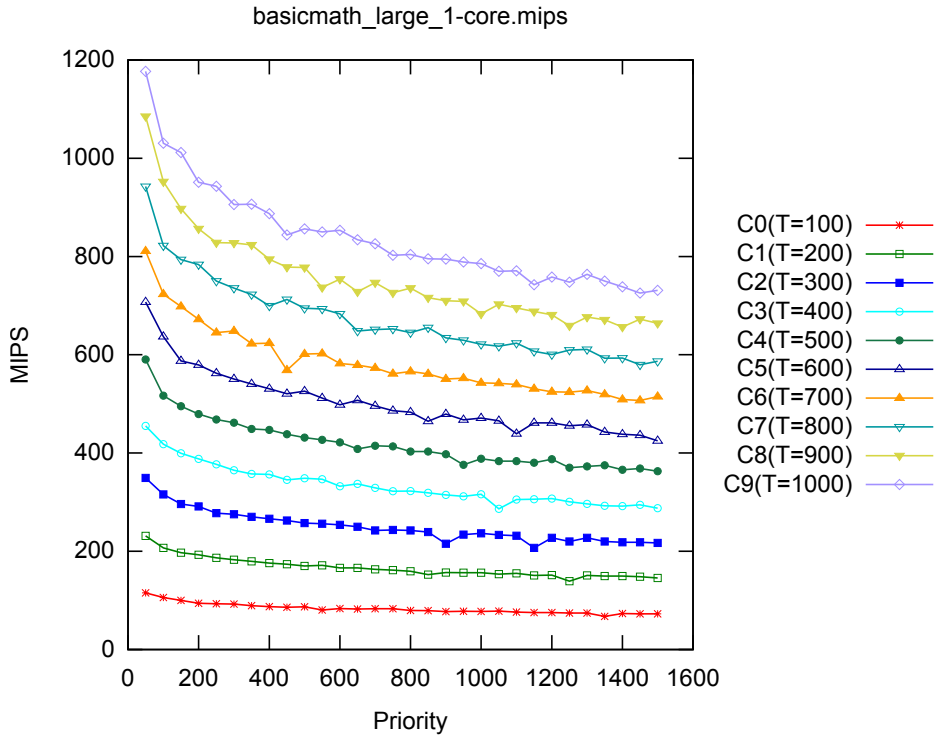


Figura 144: Desempenho em MIPS de Basicmath (Grande)

Seguindo a mesma parametrização de ajuste e prioridade vista anteriormente, as figuras 143 e 144 ilustram os desempenhos em MCPS e MIPS da apli-

cação Basicmath, com o conjunto de entrada de tamanho grande. É notória a diferença de tempo de execução entre esta entrada maior e a anterior, e devido a esta diferença considerável, é obtido um desempenho superior quando comparando com a entrada pequena. Com picos de cerca de 600 MCPS e 1.200 MIPS, esta melhoria é atribuída ao somatório de eficiência obtido pelo modelo proposto, representando uma melhoria de 632 e 1.905 vezes, respectivamente, com relação ao desempenho do ISS que é de 0,95 MCPS e 0,63 MIPS.

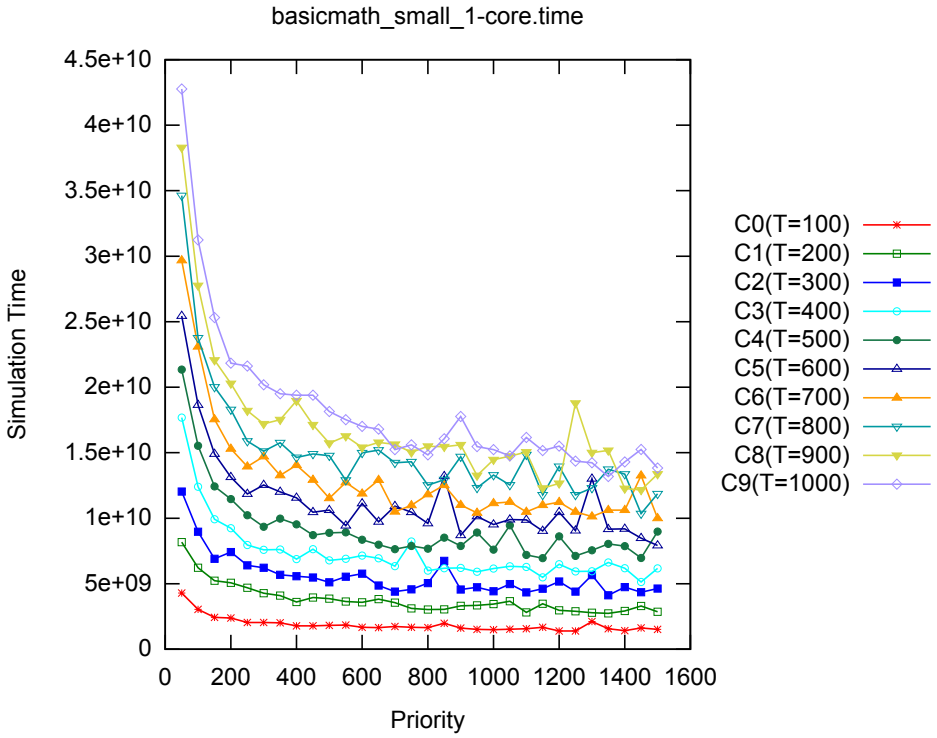


Figura 145: Tempo simulado de Basicmath (Pequeno)

Sob a perspetiva da estimativa de tempo, a simulação do Basicmath com entrada pequena tem suas curvas de tempo plotadas na figura 145. O comportamento previsto mais uma vez é confirmado, mesmo se observando os efeitos do não determinismo que extrapolam os mecanismos de filtragem propostos. No modelo ISS, foram executado um tempo total simulado de 2,292754024e+12 picosegundos. Buscando equipar o comportamento observado no ISS, são calibrados parâmetros de ajuste de valor 141.885 e de prioridade com valor 862. Com esta parametrização, foi obtido um tempo estimado de 2,231540934242e+12 picosegundos em 87 simulações, com erro relativo ao ISS de 2,66% e desvio padrão de 1,51530475222e+11 picosegundos (±6,79%). Além disto, foi obtido um desempenho de 48.410 MCPS e 96.819 MIPS que representam uma melhoria

de cerca de 53.046 e 156.945 vezes, respectivamente, quando comparado ao ISS que obteve 0,91 MCPS e 0,62 MIPS.

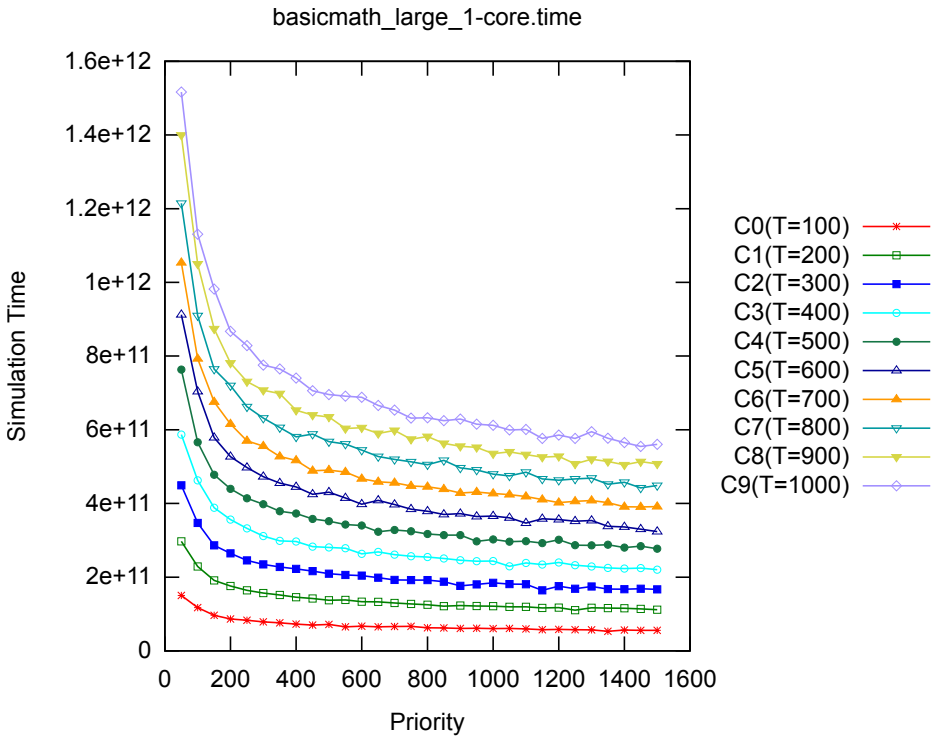


Figura 146: Tempo simulado de Basicmath (Grande)

Continuando a análise do tempo simulado estimado, é encontrada na figura 146 as estimativas geradas para a aplicação Basicmath com entrada de tamanho grande. Na simulação de referência realizada no ISS foram obtidos um tempo simulado total de 2,6641672232e+13 picosegundos, com um desempenho de 0,95 MCPS e 0,69 MIPS. Observando o modelo proposto, é desejado que o comportamento seja o mais próximo possível do observado no ISS e apresente o melhor desempenho possível. Com a calibração dos parâmetros de ajuste para 45.450 e com prioridade de 600. A estimativa de tempo simulado obtida foi de 2,8117499366357e+13 picosegundos em 5 simulações, com desempenho de 17.665 MCPS e 35.311 MIPS. A melhoria de velocidade de simulação atingida foi de cerca de 18.673 e 56.049 vezes, respectivamente, com taxa de erro de aproximadamente de 5,53% e desvio padrão de 7,7697795728e+10 picosegundos (±0,27%), considerando o modelo ISS como referência.

### B.1.2 Bitcount

No contexto automotivo, a aplicação Bitcount realiza uma série de manipulações em nível de bit em estruturas de dados. Estas manipulações incluem a contagem do número de bits em um vetor de inteiros, através da utilização de cinco métodos: contador com laço otimizado de 1 bit, contagem recursiva de bits por nibbles, contagem não recursiva de bits por bytes usando uma tabela de consulta e operações de deslocamento e contagem de bits. Mais uma vez, estas operações de manipulação de bits são relevantes em aplicações automotivas, como tratamento de sinais de sensores e ajuste de sinalizadores de informações. Os vetores de teste fornecidos são vetores de inteiros com quantidades iguais de bits zeros e uns.

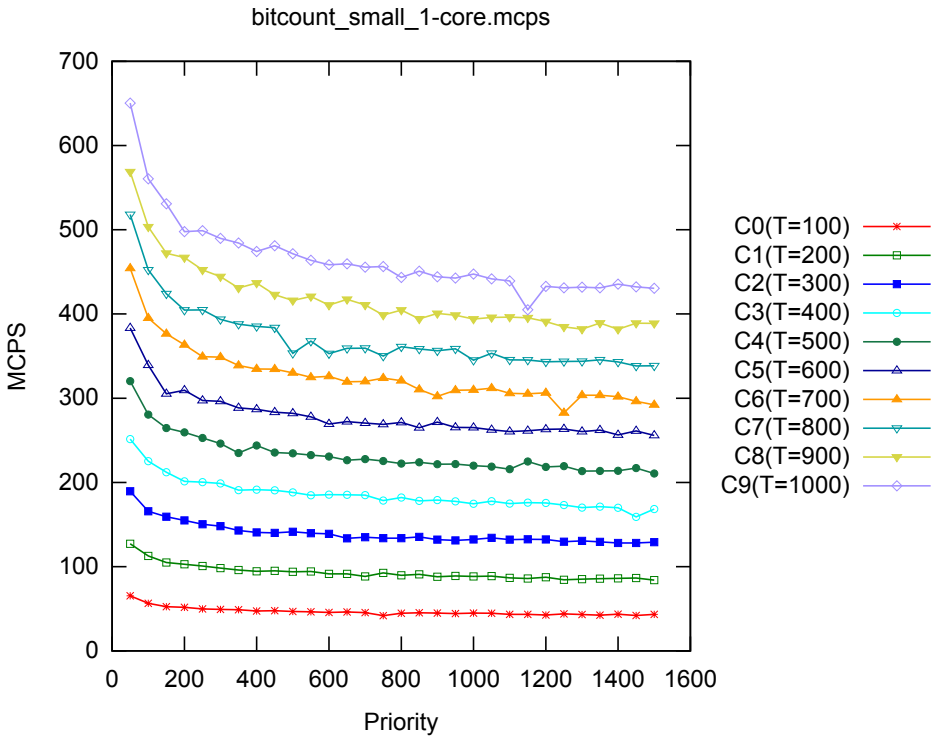


Figura 147: Desempenho em MCPS de Bitcount (Pequeno)

Utilizando o vetor de teste pequeno, a aplicação Bitcount apresenta um pico de desempenho de cerca de 650 MCPS e 1.300 MIPS, sendo ilustrados pelas figuras 147 e 148. Esta simulação segue a mesma parametrização já observada em outros exemplos, com 10 curvas de ajuste (C0 até C9), com valores de 100 até 1.000, com intervalo de valor 100, e a priorização, vista no eixo horizontal, são gerados parâmetros com valor de 50 até 1.500, com intervalos de tamanho 50. Nesta configuração de parâmetros, foi atingida

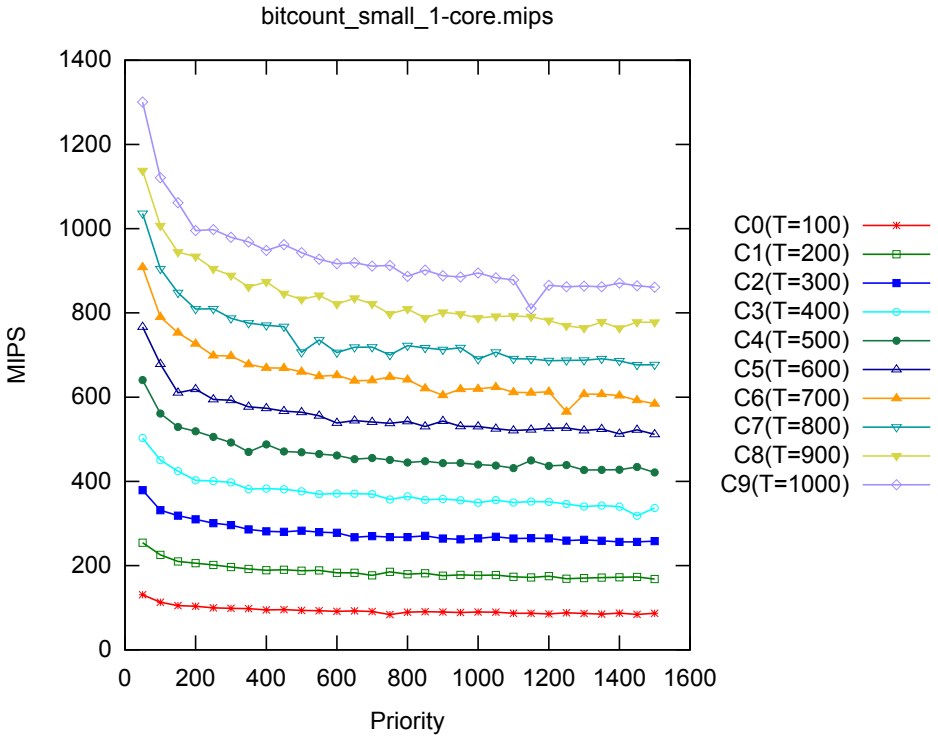


Figura 148: Desempenho em MIPS de Bitcount (Pequeno)

uma melhoria de desempenho de 492 e 1.884 vezes, considerando as métricas MCPS e MIPS, respectivamente, uma vez que o ISS apresentou desempenho de 1,32 MCPS e 0,69 MIPS.

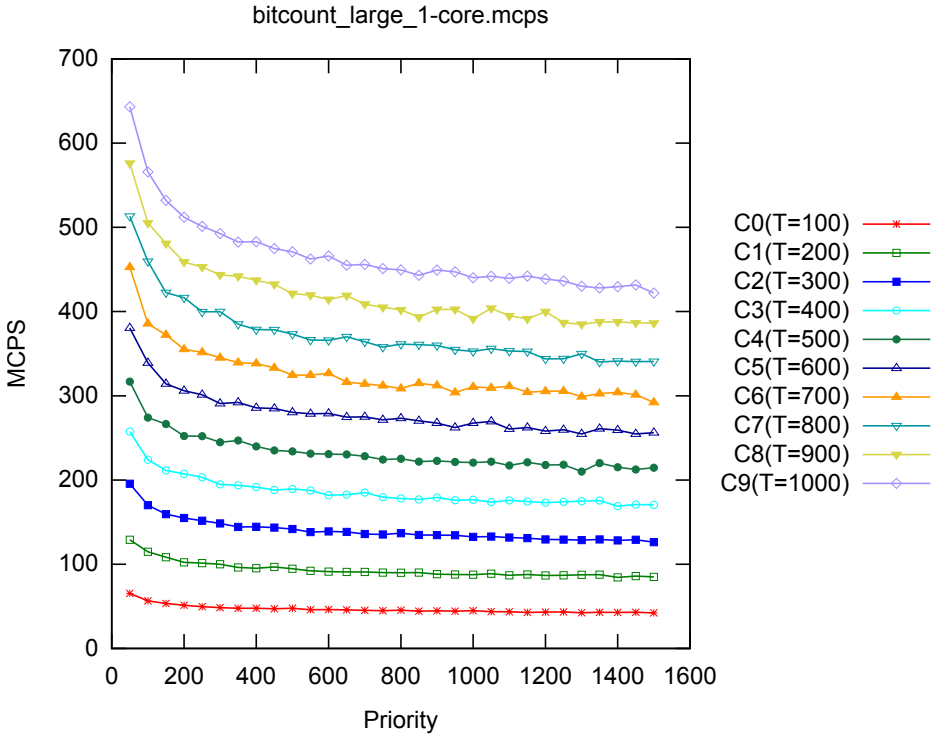


Figura 149: Desempenho em MCPS de Bitcount (Grande)

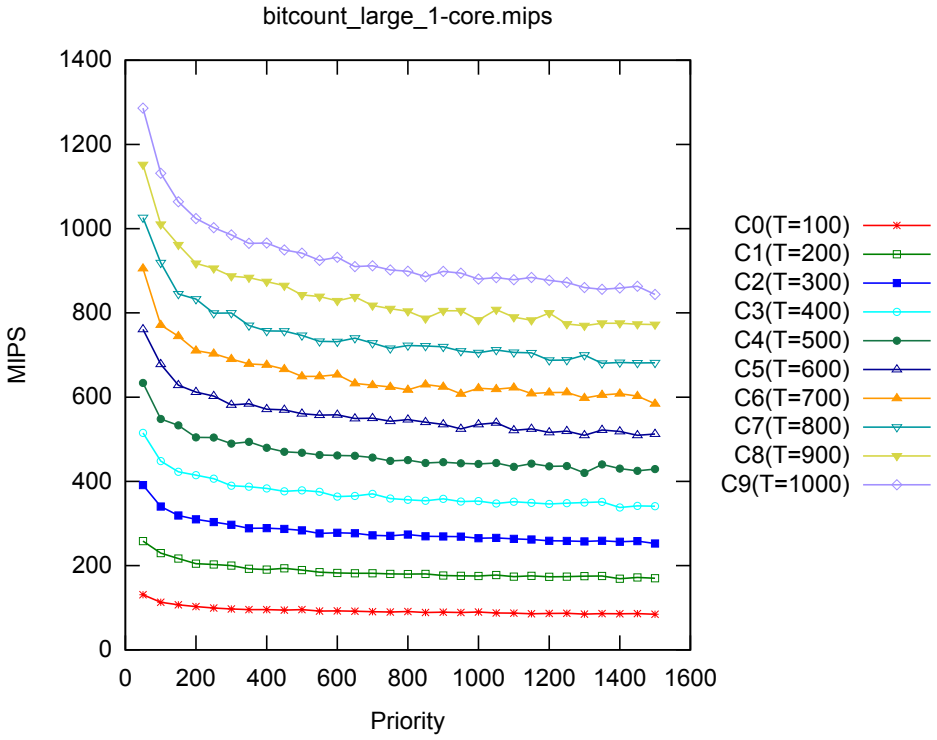


Figura 150: Desempenho em MIPS de Bitcount (Grande)

Nas figuras 149 e 150, são ilustrados os gráficos de desempenho do Bitcount para um vetor de entrada de tamanho grande. Apresentando comportamento bem similar a entrada pequena, obteve um pico de desempenho de cerca de 650 MCPS e 1.300 MIPS, considerando a mesma parametrização adotada no caso anterior. Nesta entrada, o modelo ISS obteve um desempenho de 1,33 MCPS e 0,70 MIPS, portanto a melhoria de desempenho oferecida pelo modelo proposto foi de cerca de 489 e 1.857 vezes, respectivamente.

Focando a atenção na estimativa de tempo, vista na figura 151, para a entrada pequena do Bitcount, é observado um comportamento bem previsível, considerando os parâmetros e intervalos já descritos anteriormente. Verificando o relatório de informações gerado pelo modelo ISS, foi executado um tempo simulado de 1,9871294e+11 picosegundos, atingindo desempenho de 1,32 MCPS e 0,69 MIPS. Para realização de uma simulação equivalente no modelo proposto, são calibrados os parâmetros de ajuste de valor 644 e de prioridade de valor 783, atingindo um tempo simulado de 2,0863138812e+11 picosegundos. Neste exemplo foi atingido um desempenho de 270,20 MCPS e 540,40 MIPS pelo modelo proposto, com um erro relativo de 4,99% e desvio padrão de 2,304584854e+9 picosegundos (±1,10%), considerando o ISS como referência com 0% de erro. A melhoria de desempenho obtida foi de aproxi-

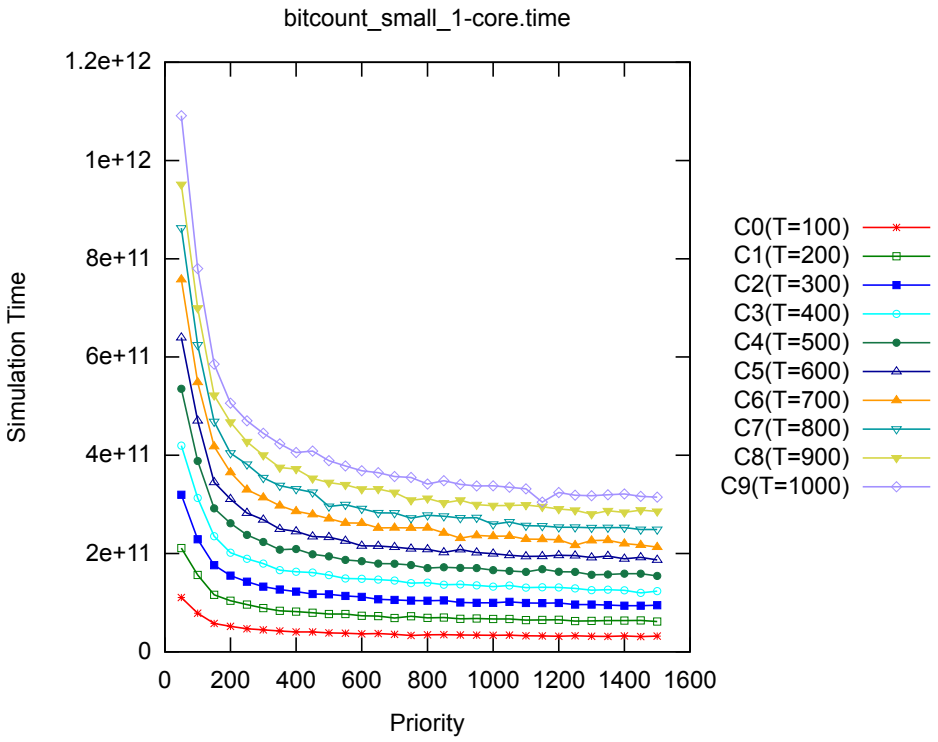


Figura 151: Tempo simulado de Bitcount (Pequeno)

madamente 205 e 780 vezes, respectivamente, para equiparar o comportamento obtido no ISS.

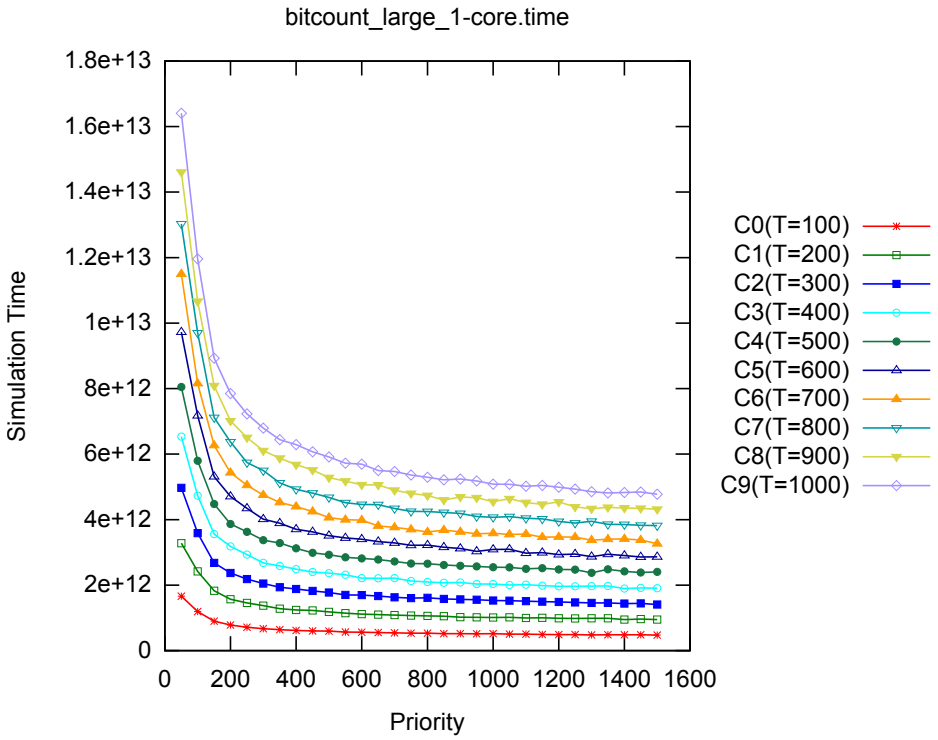


Figura 152: Tempo simulado de Bitcount (Grande)

Ainda analisando as estimativas de tempo simulado, pode-se observar na

figura 152 os resultados obtidos pelo Bitcount usando entrada de tamanho grande. Com resultados muito similares aos obtidos aos anteriores, o modelo ISS gerou um tempo simulado de 2984113301e+12 picosegundos, com desempenho de simulação de 1,33 MCPS e 0,70 MIPS. Para obter um comportamento equivalente no modelo proposto, são calibrados os parâmetros de ajuste com valor 627 e prioridade de valor 920, obtendo um tempo simulado de 3,032441055604e+12 picosegundos e desempenho de 262,20 MCPS e 524,40 MIPS em 3 simulações. A melhoria obtida com relação ao modelo ISS foi de cerca de 198 e 750 vezes, respectivamente, com erro relativo de aproximadamente 1,61% e desvio padrão de 1,079028518e+10 picosegundos (±0,35%).

### B.1.3 Quicksort

No contexto automotivo, o algoritmo Quicksort (63) é aplicado para ordenar um vetor de grande tamanho contendo cadeias de caracteres, adotando o critério de ordenação ascendente. Como já foi dito, com uma estrutura ordenada é possível realização de priorização, uma vez que os dados estão organizados, além melhorar o entendimento de determinadas saídas geradas para o usuário do sistema. Os vetores de teste fornecidos são listagem de palavras (entrada pequena) e tuplas representando pontos de informação (entrada grande).

Nas figuras 153 e 154 são ilustrados o desempenho do algoritmo de ordenação Quicksort com um tamanho de entrada pequeno. Utilizando a mesma parametrização padrão de ajuste com 10 curvas (C0 até C9) com valores de 100 até 1.000, com intervalos de tamanho 100, e prioridade entre 50 e 1.500, com passos de tamanho 50. Foi obtido um pico de desempenho em torno de 500 MCPS e 1.000 MIPS, representando uma melhoria de 532 e 1.923 vezes no desempenho do ISS que foi de 0,94 MCPS e 0,52 MIPS. Apesar de ser uma aplicação como outra qualquer, existe uma peculiaridade nesta aplicação: o algoritmo é implementado pela biblioteca padrão de C. O impacto desta característica está no fato do trabalho proposto não ser capaz de instrumentar o código utilizado, causando este comportamento ruidoso e que foge, em grande parte do tempo, ao comportamento teórico previsto. Mesmo com esta característica, é possível realizar a simulação do código, uma vez que existe um código acessório necessário para realização de operações de lei-

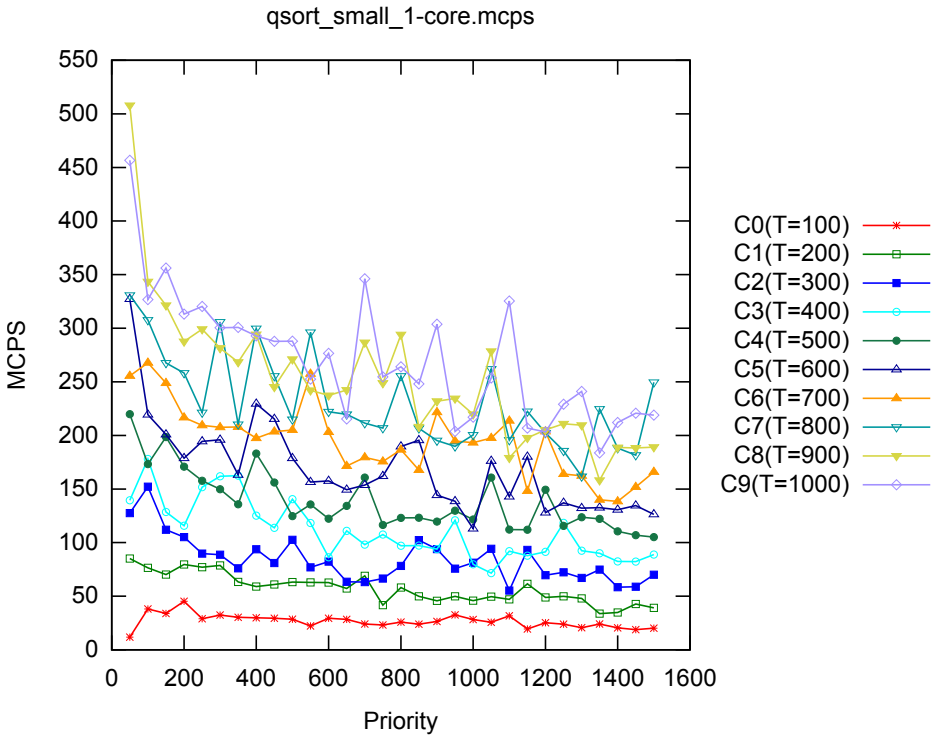


Figura 153: Desempenho em MCPS de Quicksort (Pequeno)

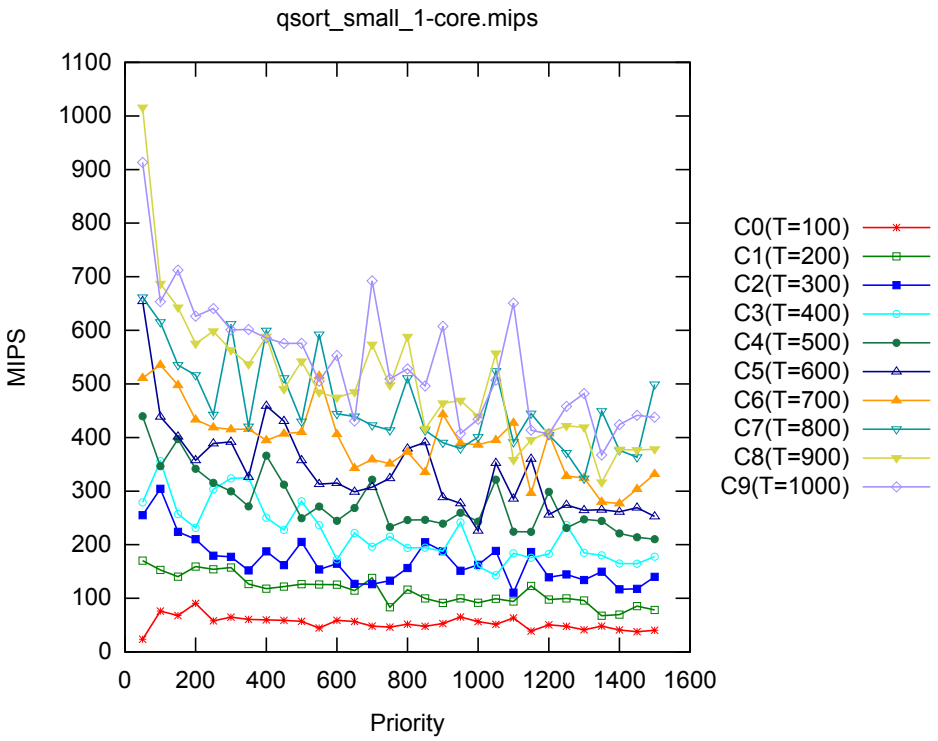


Figura 154: Desempenho em MIPS de Quicksort (Pequeno)

tura de arquivo e exibição do resultado na tela do usuário.

Seguindo o mesmo comportamento da entrada pequena, a simulação com a entrada grande obteve o mesmo resultado ruidoso de desempenho

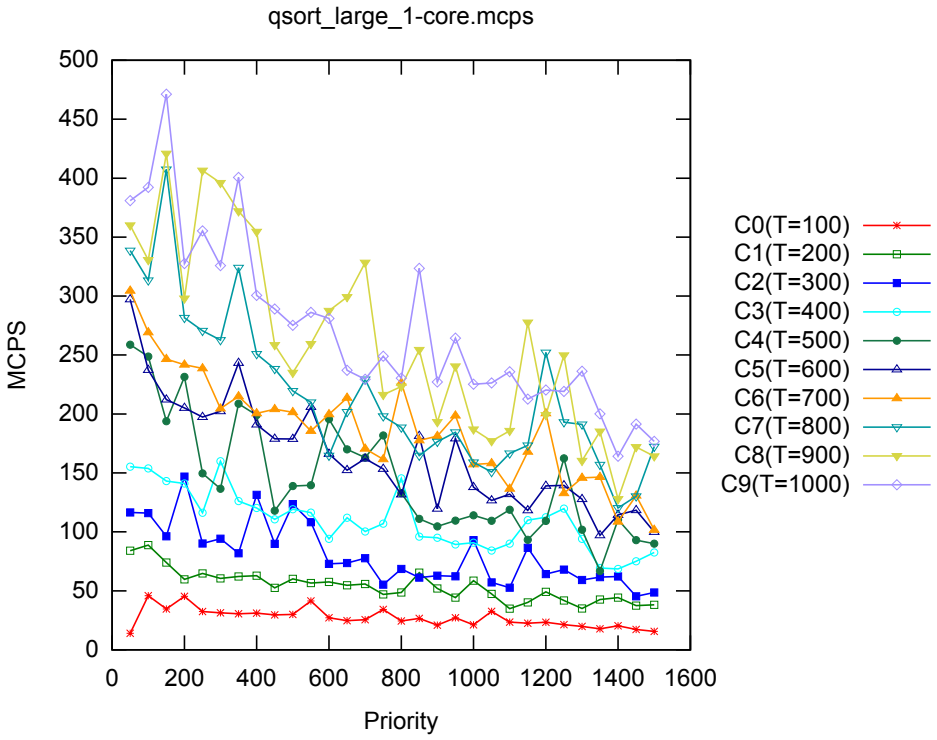


Figura 155: Desempenho em MCPS de Quicksort (Grande)

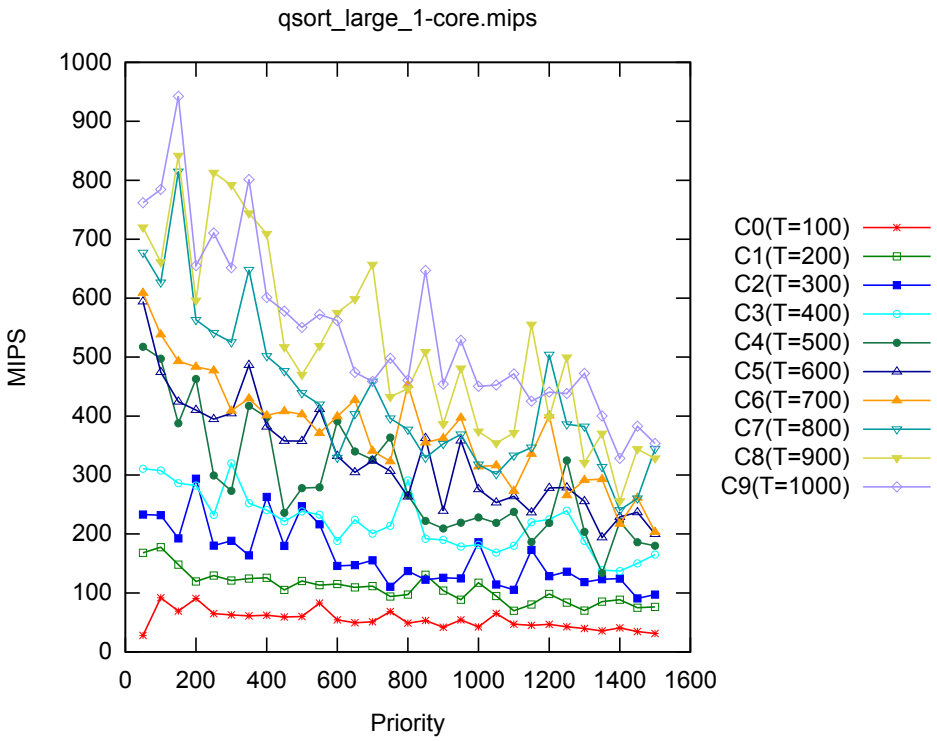


Figura 156: Desempenho em MIPS de Quicksort (Grande)

que pode ser visualizado nas figuras 155 e 156. Foi utilizada a mesma parametrização do exemplo anterior, mas foi atingido um pico de desempenho de cerca de 450 MCPS e 950 MIPS, promovendo uma melhoria de 459 e 1.439 ve-

zes no desempenho, quando comparado ao ISS com 0,98 MCPS e 0,66 MIPS. Como o tamanho de entrada foi ampliado, os efeitos da execução do código não instrumentado foi maximizado, tornando os resultados menos previsíveis.

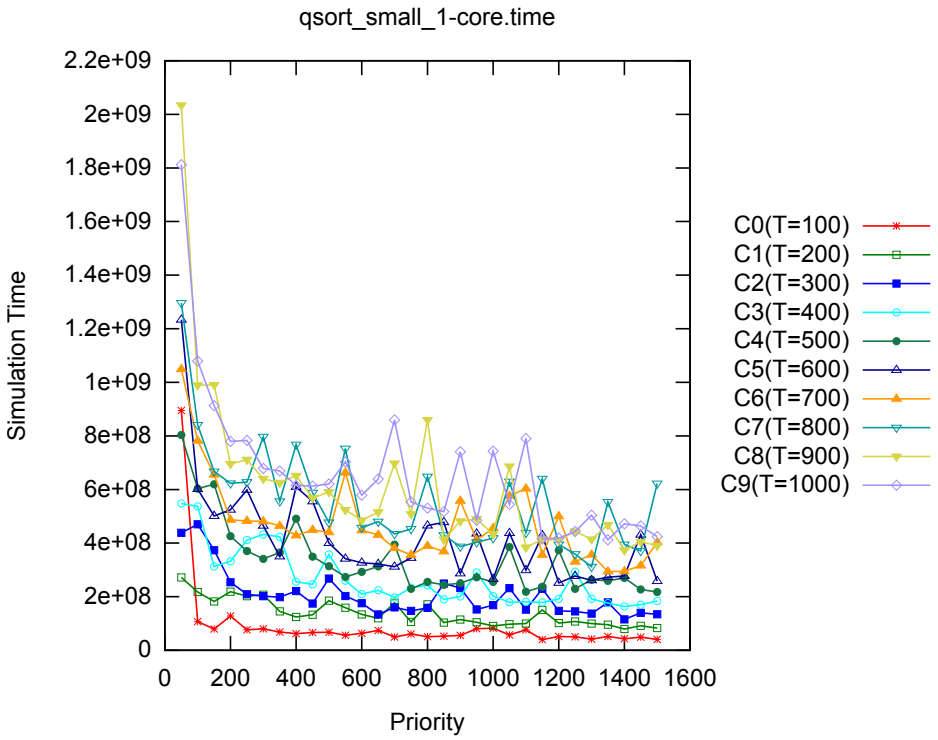


Figura 157: Tempo simulado de Quicksort (Pequeno)

Observando as estimativas de tempo simulado da figura 157, para entrada pequena do Quicksort, é possível observar que a medida que a priorização aumenta, a granularidade aumenta, reduzindo o controle do simulador sobre o tempo simulado. Como já foi dito, grande parte do código está implementado na biblioteca padrão C e desta forma não pode ser instrumentado, causando estes efeitos não previstos pelo modelo teórico. No modelo ISS, foi obtido um tempo simulado de 7,308481e+9 picosegundos, atingindo um desempenho de 0,94 MCPS e 0,52 MIPS. Para obter um comportamento equivalente no modelo proposto, são calibrados parâmetros de ajuste com valor 12.326 e prioridade de valor 607, atingindo um tempo simulado de 8,572232539e+9 picosegundos em 1.311 simulações. O desempenho obtido foi de 3.242 MCPS e 6.484 MIPS, o que representa uma melhoria de desempenho de aproximadamente 3.455 e 12.468 vezes, respectivamente, quando comparando ao desempenho do ISS, com uma taxa de erro relativo de cerca de 17,29% e desvio padrão de 1,405721992e+9 picosegundos (±16,39%).

Realizando estimativas de tempo simulado para o Quicksort com uma en-

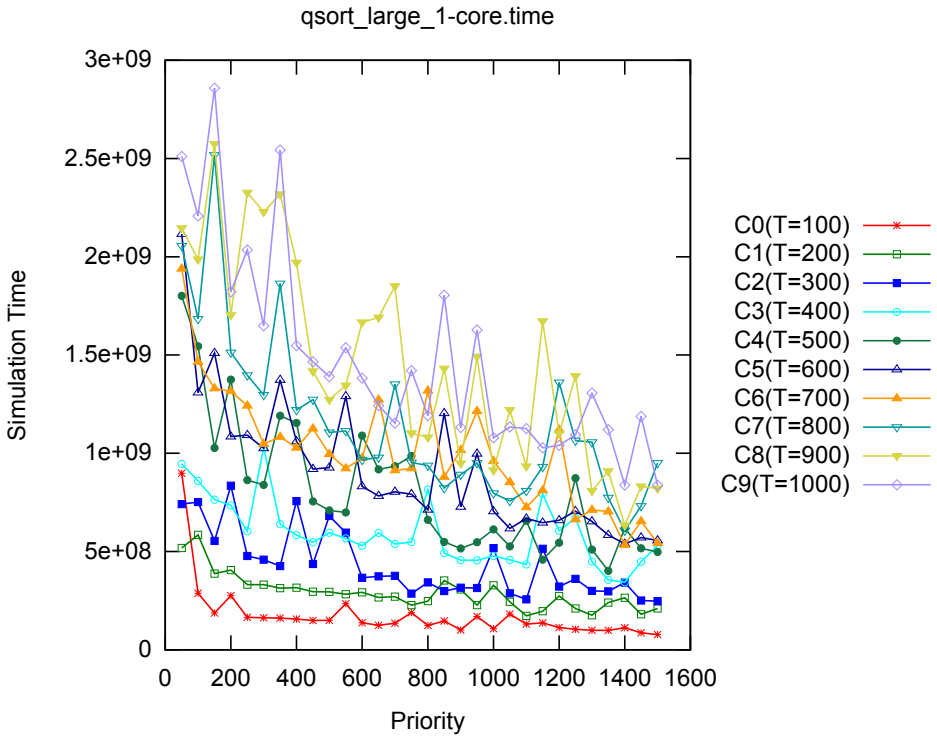


Figura 158: Tempo simulado de Quicksort (Grande)

trada grande, é gerada a figura 158. Mais uma vez o comportamento teórico previsto não é obtido pela não instrumentação do código do Quicksort que está embutido na biblioteca padrão de C. Apesar disto, é possível realizar a simulação e obter estimativas que podem ser utilizadas para desenvolvimento e análise do comportamento obtido. Na simulação com ISS foi obtido um tempo simulado de 5,9064008e+10 picosegundos, com desempenho de 0,98 MCPS e 0,66 MIPS. No modelo proposto é desejado que o comportamento mais próximo possível seja obtido e são calibrados parâmetros de ajuste com valor 46.004 e prioridade de valor 627, obtendo um tempo simulado de 6,7565543838e+10 picosegundos em 714 simulações. Com um desempenho de 11.619 MCPS e 23.238 MIPS, foi obtida uma melhoria de aproximadamente 11.881 e 35.231 vezes, respectivamente, com erro associado de 14,39% e desvio padrão de 1,0098460706e+10 picosegundos (±14,94%), quando comparando aos resultados do ISS.

### B.1.4 Susan

Com a crescente demanda por automatização em sistemas automotivos, o processamento de imagem passa a ter relevância como assistente de dire-

ção para o condutor do veículo. Por este motivo, o pacote MiBench possui uma aplicação chamada Susan que é capaz de realizar reconhecimento de imagem, como pedestres ou obstáculos na pista. Com este suporte adicional proporcionado por esta visão artificial, é esperado que o condutor do veículo e seus passageiros tenham mais segurança ao utilizarem o veículo. O aplicativo Susan realiza o processo de reconhecimento de imagens através da detecção de cantos e bordas da imagem, tendo sua utilização original em imagens de ressonância magnética de cérebros.

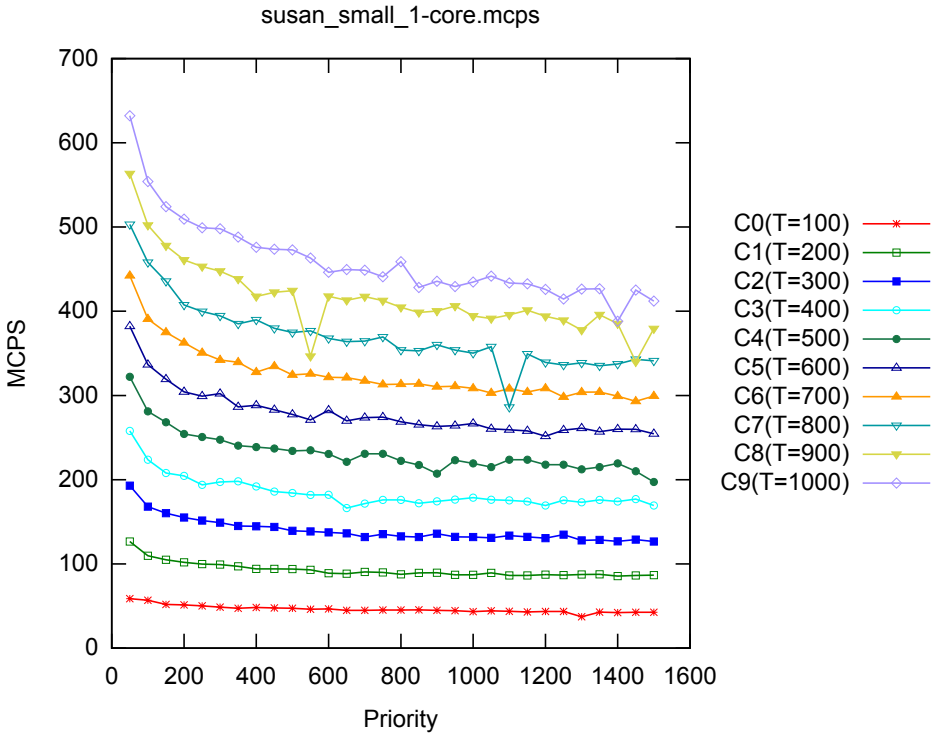


Figura 159: Desempenho em MCPS de Susan (Pequeno)

Nas figuras 159 e 160, são ilustradas as análises de desempenho da aplicação Susan com as curvas de ajuste C0 até C9, com valores de 100 até 1.000, com intervalos de tamanho 100. No eixo horizontal, a priorização é definida entre valores de 50 até 1.500, com passos de tamanho 50, para cada uma das 10 curvas de ajuste geradas. Nesta parametrização foi obtido um pico de desempenho de cerca de 600 MCPS e 1.200 MIPS, com melhoria de desempenho de 577 e 1.905 vezes, respectivamente, quando comparado ao ISS que obteve 1,04 MCPS e 0,63 MIPS.

Ainda focando na análise de desempenho, mas utilizando uma entrada de tamanho grande, a aplicação Susan obteve um comportamento bem similar e com poucos efeitos do não determinismo da execução nativa, como pode

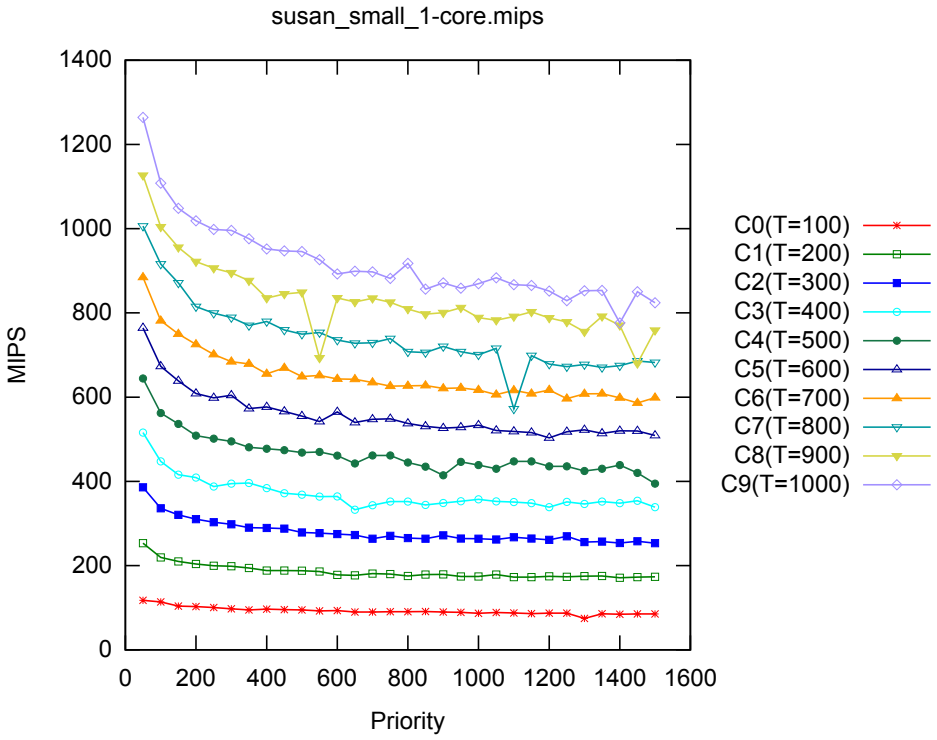


Figura 160: Desempenho em MIPS de Susan (Pequeno)

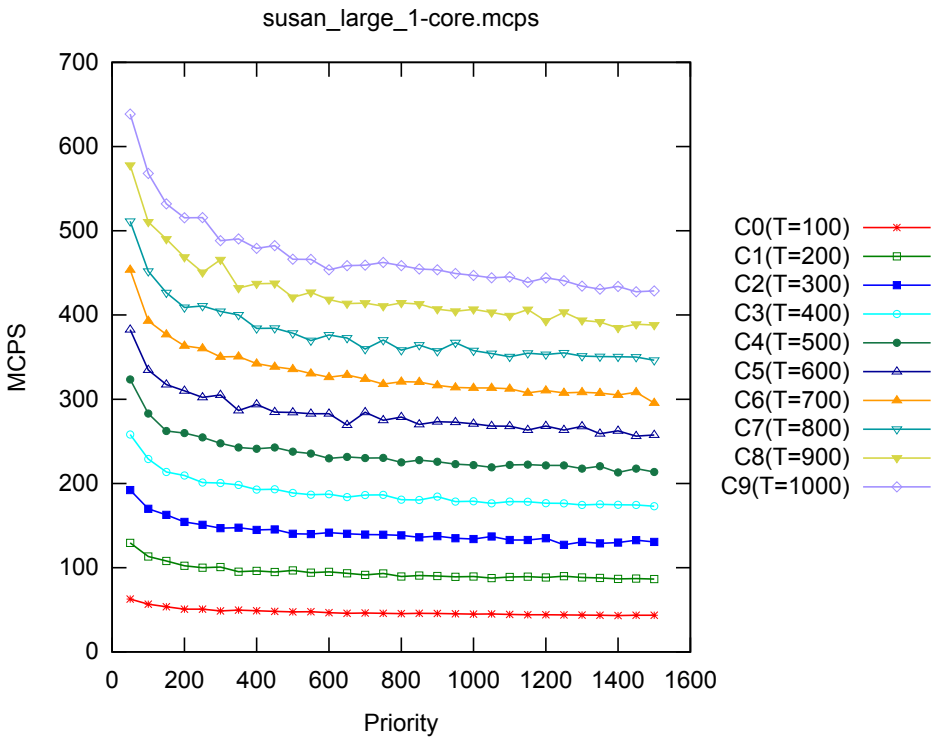


Figura 161: Desempenho em MCPS de Susan (Grande)

ser visualizado nas figuras 161 e 162. Aplicando a mesma parametrização do exemplo anterior, foi obtido também um pico de desempenho de cerca de 650 MCPS e 1.300 MIPS, proporcionando um aumento de desempenho de 608

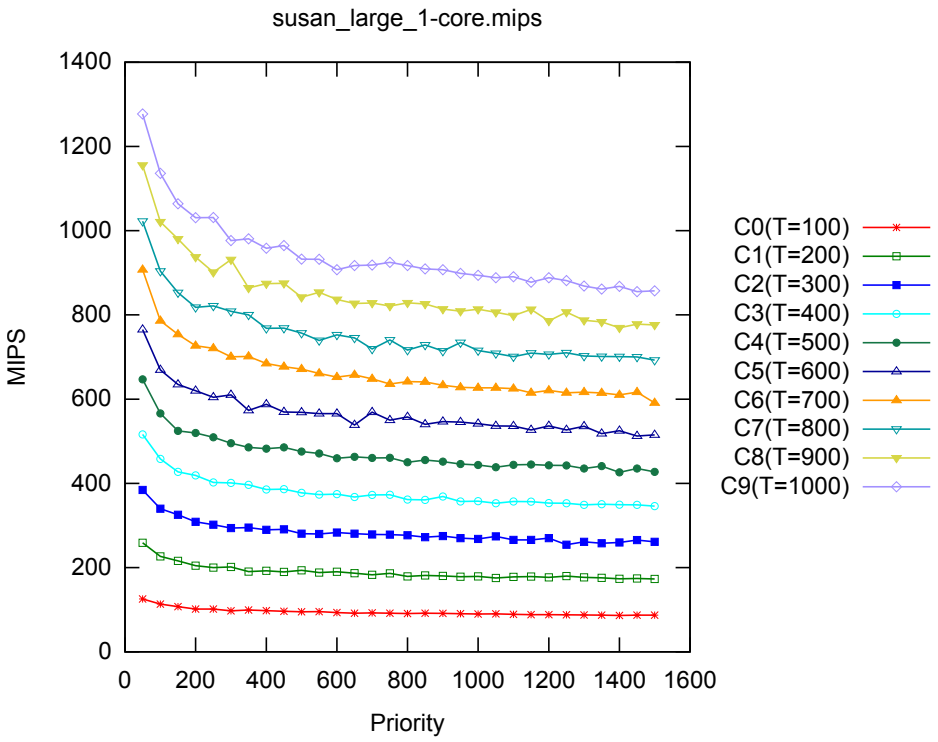


Figura 162: Desempenho em MIPS de Susan (Grande)

e 2.031 vezes com relação ao ISS com 1,07 MCPS e 0,64 MIPS de desempenho.

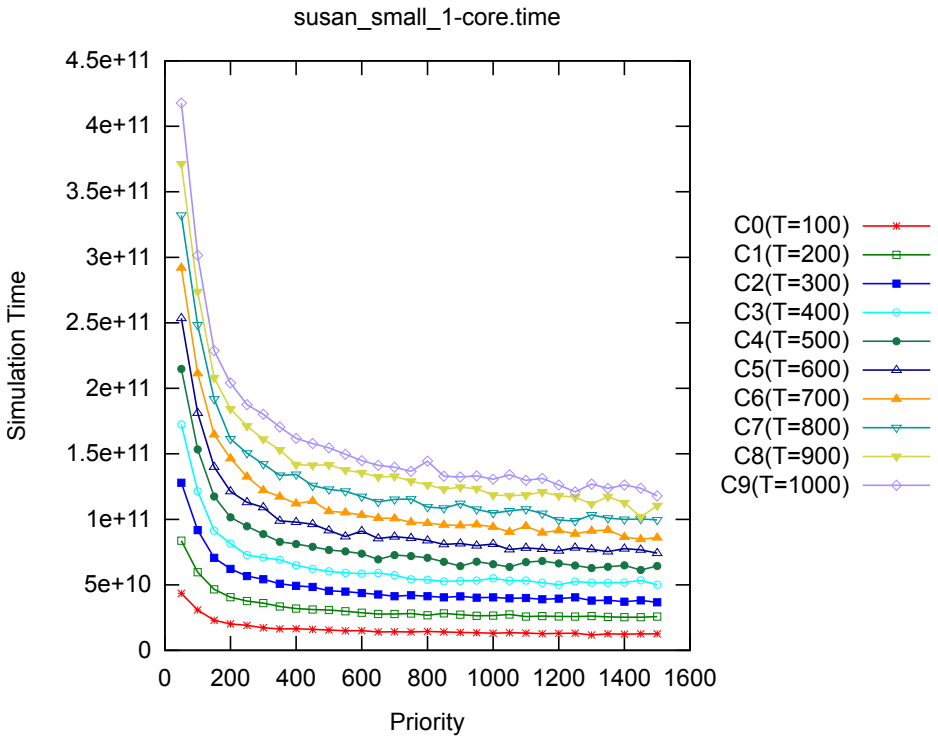


Figura 163: Tempo simulado de Susan (Pequeno)

Observando a figura 163, é possível visualizar as estimativas de tempo realizadas pelo modelo proposto e como seu comportamento é fiel ao seu modelo

teórico, conseguindo controlar com muito sucesso os efeitos aleatórios causados pela simulação nativa. Na simulação com o modelo ISS foi obtido um tempo simulado de 2,78697783e+11 picosegundos, atingindo um desempenho de 1,04 MCPS e 0,63 MIPS. Conforme o objetivo de abstração de processamento, são calibrados os parâmetros de ajuste com o valor 2.244 e de prioridade com o valor 942, para se obter um comportamento o mais próximo possível do observado no ISS. Esta escolha de parâmetros é sempre feita buscando maximizar o desempenho obtido e minimizar a taxa de erro das estimativas de tempo geradas, gerando uma estimativa de tempo de 2,77235386022e+11 picosegundos com 13 simulações. Foi obtido um desempenho de 914,44 MCPS e 1.829 MIPS, que representa um melhoria de 876 e 2.893 vezes do desempenho, respectivamente, apresentado pelo modelo ISS, exibindo uma taxa de erro relativo de aproximadamente 0,52% e desvio padrão de 5,761452051e+9 picosegundos (±2,07%).

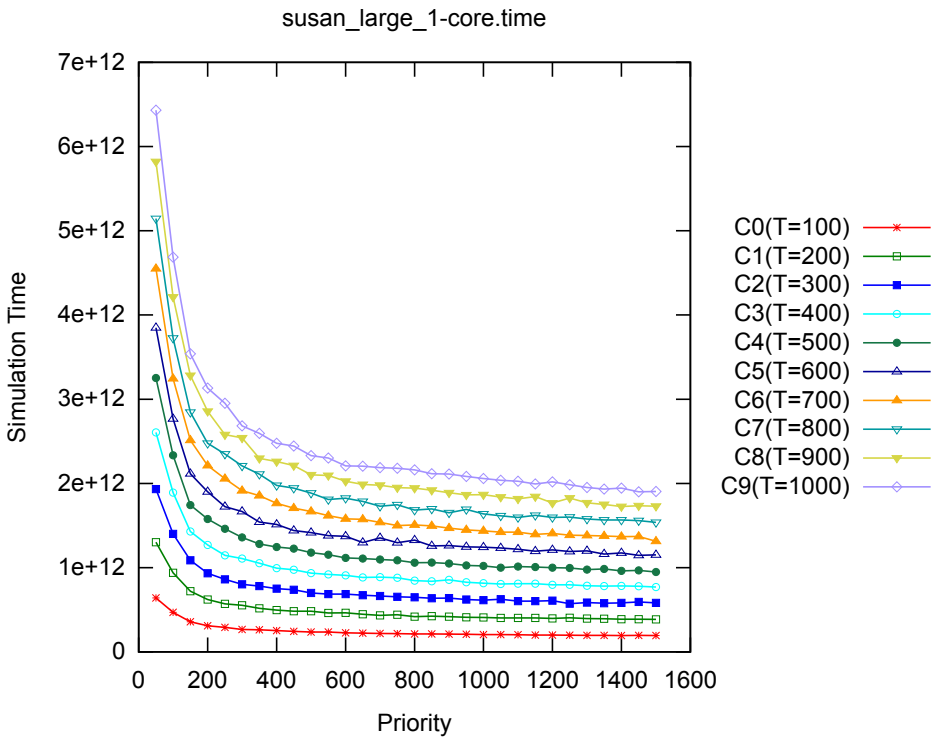


Figura 164: Tempo simulado de Susan (Grande)

Analisando as estimativas de tempo simulado para a entrada grande da aplicação sua, ilustrada na figura 164, é observado mais uma vez o comportamento previso pelo modelo teórico, mas com os efeitos do não determinismo da simulação nativa. No relatório de execução do modelo ISS foi obtido um tempo simulado de 4,027932887e+12 picosegundos, com desempenho de

1,07 MCPS e 0,64 MIPS. Buscando obter o mesmo comportamento exibido no ISS são calibrados os parâmetros de ajuste com valor de 2.137 e prioridade com valor de 1.057, gerando uma estimativa de tempo de 3,963637510367e+12 picosegundos em 5 simulações. Com estes resultados, o erro relativo ao ISS foi de cerca de 1,59% e desvio padrão de 4,035141561e+10 picosegundos (±1,01%), sendo obtido um desempenho de simulação de 873,71 MCPS e 1.747 MIPS que representa uma melhoria de 819 e 2.727 vezes do desempenho da simulação, respectivamente.

## B.2 MiBench Consumer

Neste conjunto de aplicações do MiBench são encontradas aplicações relacionadas as atividades de consumidores, como criação e exibição de imagens no formato JPEG ou na criação de conteúdo através de ferramenta de texto.

### B.2.1 JPEG Decoder

O padrão de imagem criado pelo Joint Photographic Experts Group (JPEG) (84) é um dos mais utilizados pela maioria dos sistemas modernos, e por este motivo o MiBench oferece suporte para esta aplicação tão relevante em muitos sistemas embarcados. Com esta aplicação é possível avaliar o desempenho de diversas plataformas na tarefa de decodificação de imagens, mas sem dependências tecnológicas e permitindo a avaliação da eficiência relativa da plataforma em questão. Para realização dos testes são utilizadas diferentes imagens comprimidas (tamanho pequeno e grande) para serem decodificadas e visualizadas.

Nas figuras 165 e 166, são ilustrados os desempenhos em MCPS e MIPS para decodificador JPEG com uma imagem de tamanho pequeno. Nesta parametrização foram utilizados 10 valores de ajuste (curvas C0 até C9), com valores de 100 até 1.000, com intervalos de tamanho 100, respectivamente. No eixo horizontal da prioridade, são gerados valores no intervalo de 50 até 1.500, com passos de tamanho 50, para cada uma das 10 curvas exibidas no gráfico.Nesta configuração pode ser observado eventuais efeitos do não deter-

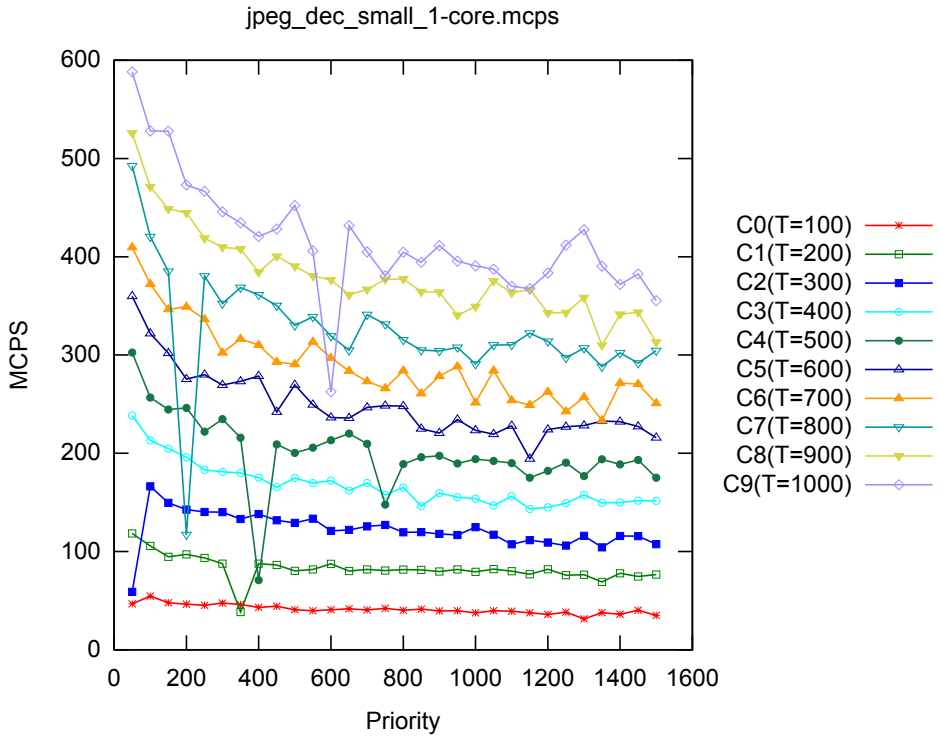


Figura 165: Desempenho em MCPS de Decodificador JPEG (Pequeno)

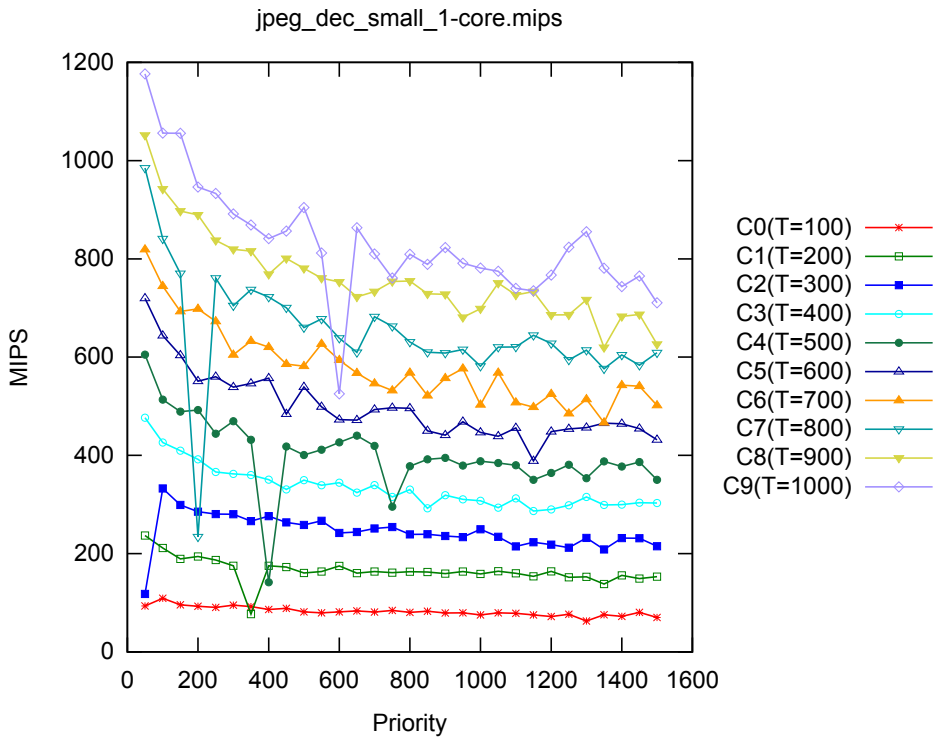


Figura 166: Desempenho em MIPS de Decodificador JPEG (Pequeno)

minismo da execução nativa, gerando momentos de descontinuidade, mas independente destes raros pontos foi atingido um pico de cerca de 600 MCPS e 1.200 MIPS com esta parametrização. Estes resultados representam uma me-

lhoria de 496 e 1.905 vezes, respectivamente, no desempenho do ISS com 1,21 MCPS e 0,63 MIPS.

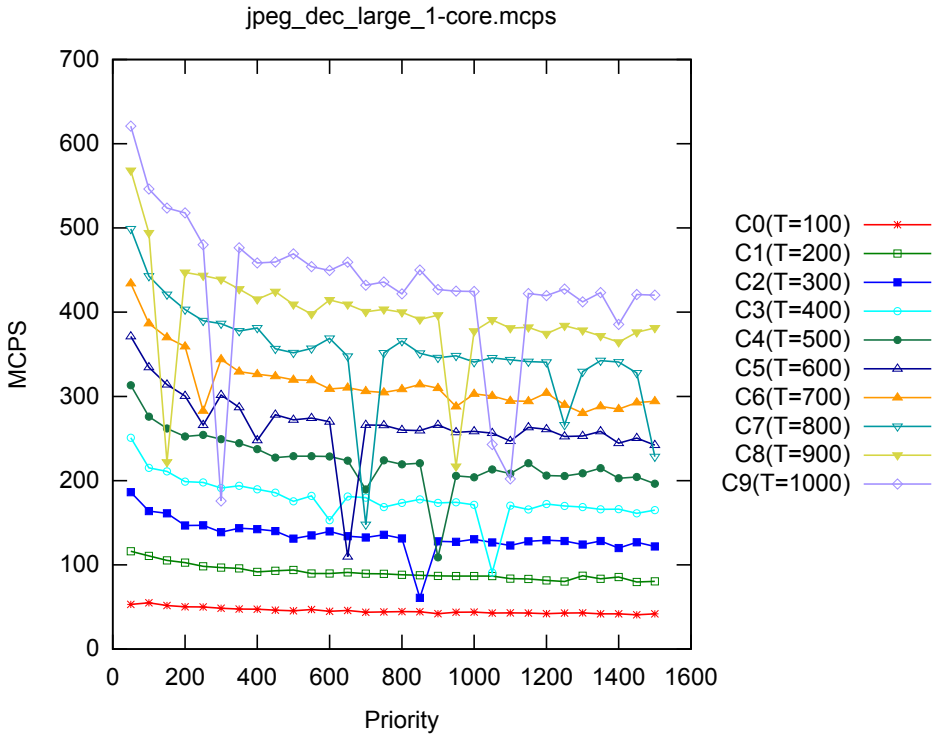


Figura 167: Desempenho em MCPS de Decodificador JPEG (Grande)

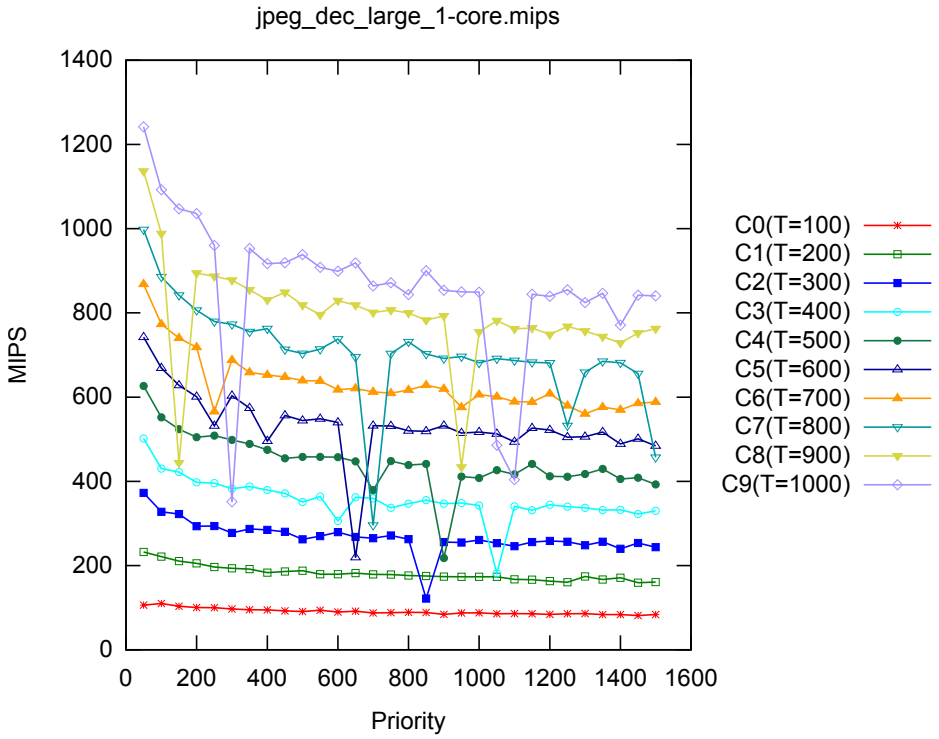


Figura 168: Desempenho em MIPS de Decodificador JPEG (Grande)

Aplicando a mesma parametrização descrita no exemplo anterior, a figura

168 ilustra o desempenho do decodificador JPEG para uma imagem de tamanho grande. Atingindo um pico de desempenho de aproximadamente 600 MCPS e 1.200 MIPS, esta parametrização obteve um comportamento bastante similar ao exemplo anterior, com uma pequena melhoria no desempenho e uma maior estabilidade com relação a execução nativa. Foi obtida nesta configuração uma melhoria de 465 e 1.818 vezes, respectivamente, do desempenho obtido pelo ISS que foi de 1,29 MCPS e 0,66 MIPS.

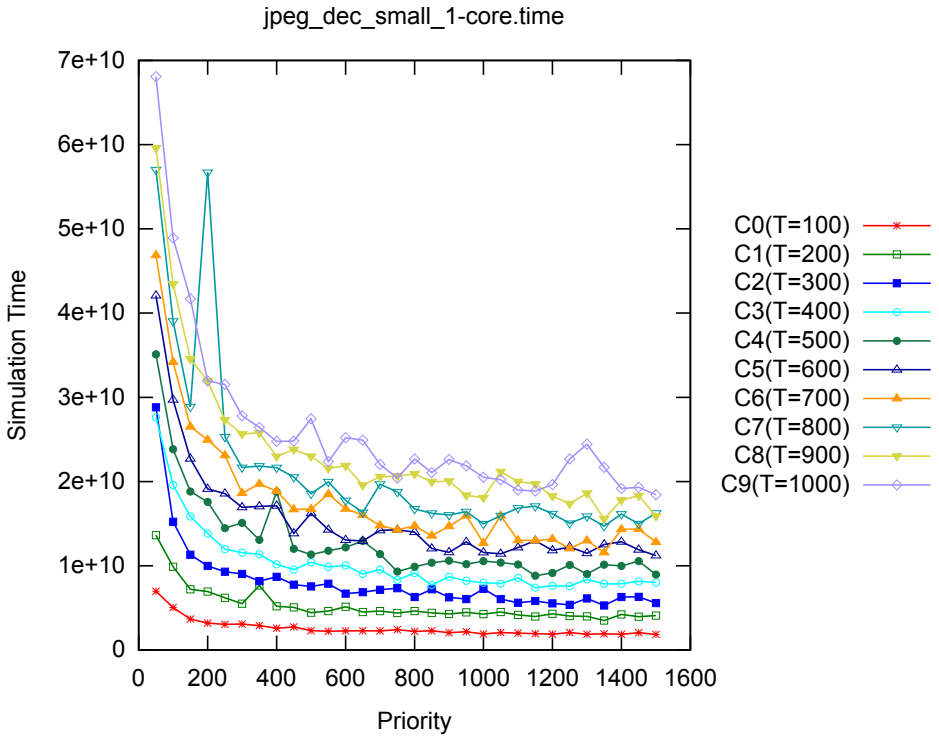


Figura 169: Tempo simulado de Decodificador JPEG (Pequeno)

Na análise das estimativas de tempo simulado, a figura 169 exibe todas as curvas geradas, aplicando a parametrização definida, e pode ser observado que o comportamento obtido corresponde a previsão teórica do modelo proposto. Na simulação com o modelo ISS foi obtido um tempo simulado de 3,3777638e+10 picosegundos, com desempenho de 1,21 MCPS e 0,63 MIPS. Como se deseja abstrair o processamento com o modelo proposto, uma consequência natural é obter um comportamento mais próximo possível do ISS que se deseja abstrair através da definição dos parâmetros de ajuste e de prioridade. Para obter o mesmo tempo simulado do ISS, são calibrados parâmetros de ajuste com valor de 1.757 e de prioridade 1.091, com o objetivo de maximizar o desempenho e minimizar o erro da estimativa, gerando um tempo simulado de 3,3511119418e+10 picosegundos com 72 simulações.

Atingindo um desempenho de 637,03 MCPS e 1.274 MIPS, o modelo proposto obteve uma melhoria de desempenho de cerca de 526 e 2.034 vezes, respectivamente, com um erro relativo de 0,78% e desvio padrão de 1,919577343e+9 picosegundos (±5,72%).

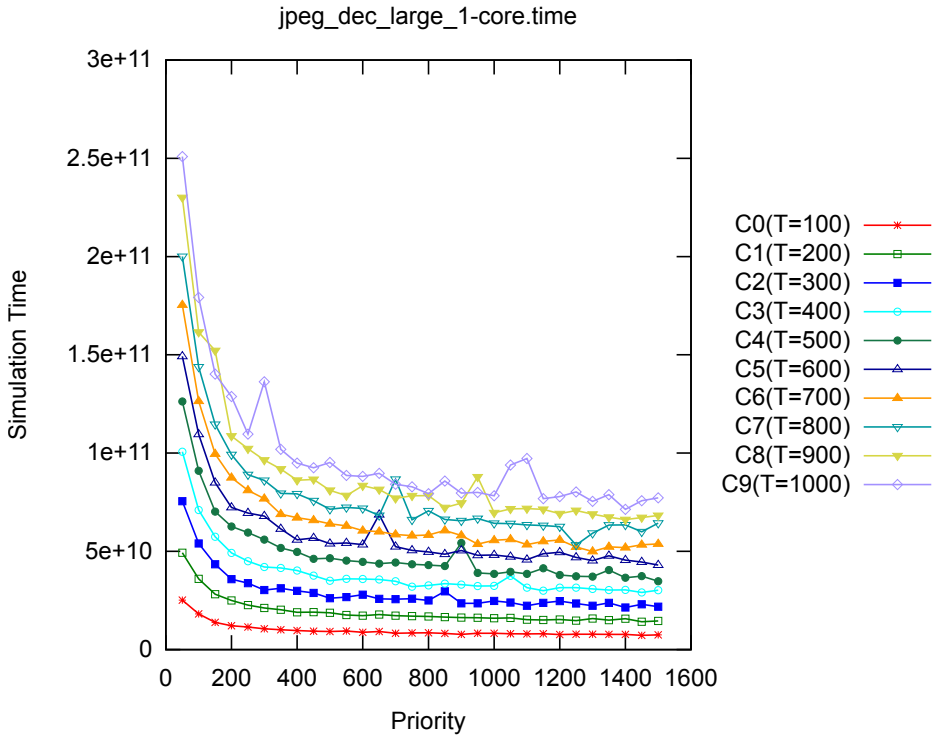


Figura 170: Tempo simulado de Decodificador JPEG (Grande)

Exibindo exatamente o comportamento esperado pelo modelo proposto, a figura 170 ilustra as estimativas de tempo simulado calculadas para a parametrização de ajuste e prioridade descritas no exemplo anterior. Na simulação realizada com ISS foi obtido um tempo simulado de 1,17903598e+11 picosegundos, com desempenho de 1,29 MCPS e 0,66 MIPS. Para equiparar este comportamento do ISS, no modelo proposto devem ser definidos valores para os parâmetros de ajuste de 1.570 e prioridade de 1.010, gerando um tempo simulado de 1,16796786866e+11 picosegundos em 38 simulações. Com um desempenho de 632,99 MCPS e 1.266 MIPS, foi atingida uma melhoria de desempenho, com relação ao ISS, de cerca de 491 e 1.914 vezes, respectivamente, com um erro associado de aproximadamente 0,93% e desvio padrão de 3,34936273e+9 picosegundos (±2,86%), considerando o ISS como referência de precisão.

### B.2.2 JPEG Encoder

O padrão JPEG realiza uma compressão com perdas da imagem digital obtida de um sensor de imagem. Por sua grande adoção em sistemas embarcados, o MiBench possui o codificador JPEG em sua base de aplicações para usuário (consumer), permitindo a avaliação do desempenho de diversas plataformas que capturam e gerenciam imagens digitais. Como entradas de teste são fornecidas diferentes imagens sem compressão para serem codificadas (uma imagem pequena e outra grande), sendo armazenadas em arquivos de saída que podem ser visualizados e comparados com as imagens originais.

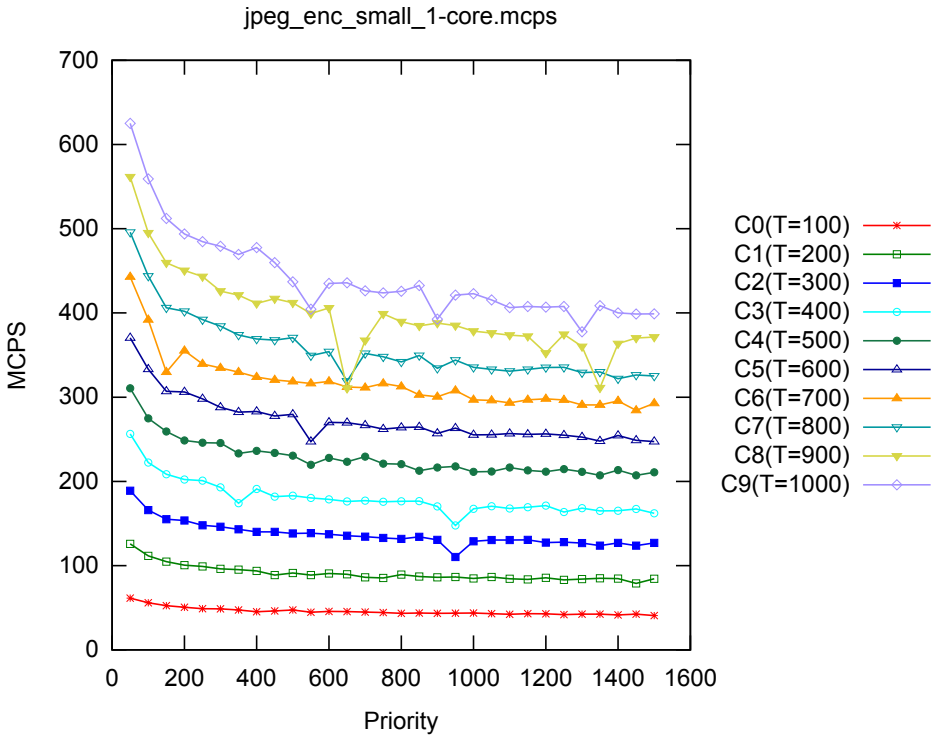


Figura 171: Desempenho em MCPS de Codificador JPEG (Pequeno)

Realizando simulações com 10 diferentes parâmetros de ajuste (curvas C0 até C9), com valores de 100 até 1.000 e intervalos de tamanho 100, juntamente com a priorização de 50 até 1.500, com passos de tamanho 50, foram obtidos os seguintes gráficos de desempenho, exibidos nas figuras 171 e 172. Adotando esta faixa de parâmetros foi obtido um desempenho máximo de aproximadamente 600 MCPS e 1.200 MIPS, considerando uma entrada de tamanho pequeno. Este desempenho do modelo proposto representa uma melhoria de 458 e 1.739 vezes, respectivamente, quando comparado ao ISS com

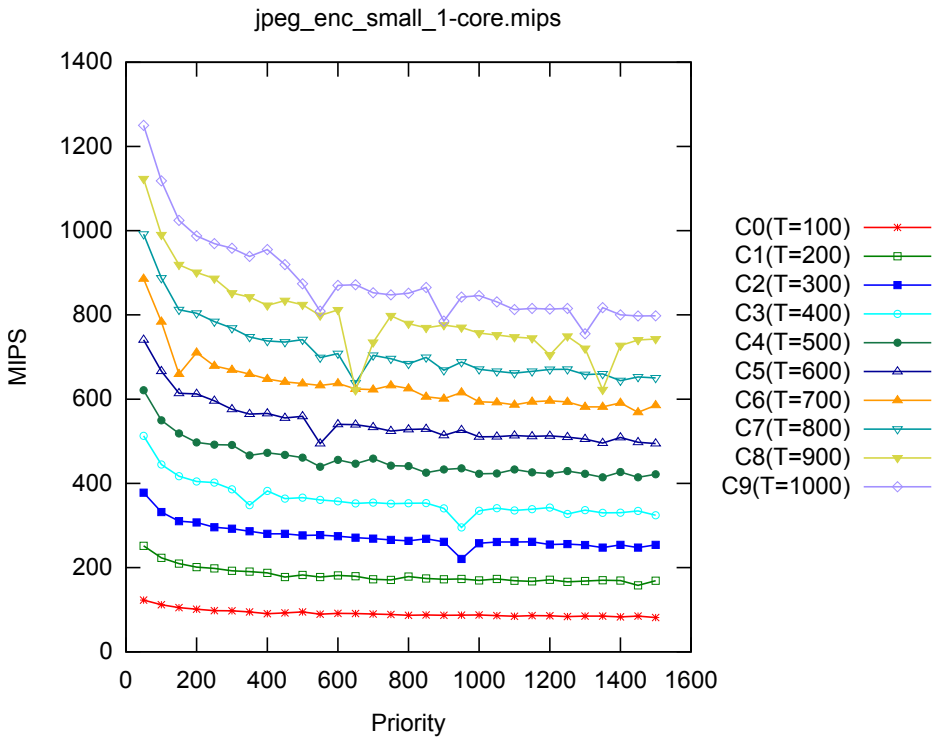


Figura 172: Desempenho em MIPS de Codificador JPEG (Pequeno)

desempenho de 1,31 MCPS e 0,69 MIPS.

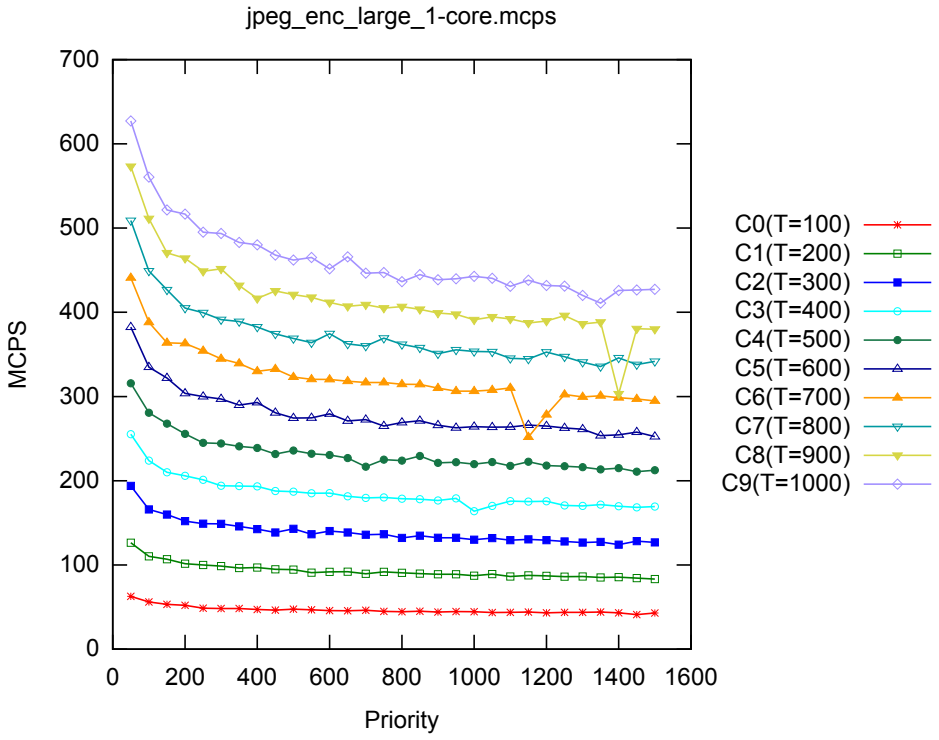


Figura 173: Desempenho em MCPS de Codificador JPEG (Grande)

Usando a mesma parametrização da entrada de tamanho pequeno, os desempenhos das simulações com entrada de tamanho grande podem ser vi-

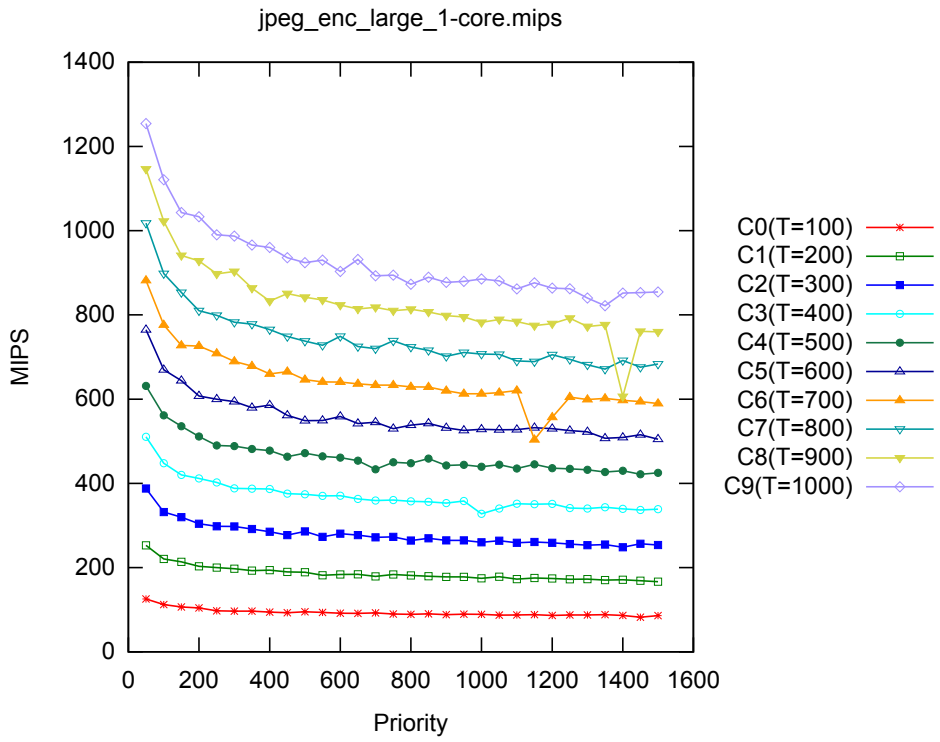


Figura 174: Desempenho em MIPS de Codificador JPEG (Grande)

sualizadas nas figuras 173 e 174. Com um topo de desempenho próximo a 600 MCPS e 1.200 MIPS, apresenta um comportamento bastante regular e muito próximo ao observado no exemplo anterior, mas também contendo efeitos do não determinismo decorrentes da execução nativa. Foi obtida uma melhoria de desempenho de 458 e 1.739 vezes, respectivamente, em relação ao ISS com 1,31 MCPS e 0,69 MIPS de desempenho.

Executando o codificador JPEG com uma entrada de tamanho pequeno, são obtidas as curvas de estimativa de tempo simulado ilustradas na figura 175. Excluindo as pequenas variações em alguns pontos, o comportamento observado nos experimentos é exatamente o previsto pelo modelo proposto, confirmando os resultados teóricos gerados. Na simulação usando ISS foi obtido um tempo simulado de 1,15859095e+11 picosegundos, apresentando um desempenho de 1,31 MCPS e 0,69 MIPS. Como é desejável obter um comportamento o mais próximo possível do ISS são calibrados parâmetros de ajuste com valor 510 e de prioridade com valor 953, para que uma estimativa de tempo próxima da obtida do ISS seja feita, buscando maximizar o desempenho e minimizar o erro gerado na aproximação. A estimativa de tempo simulado gerada foi de 1,14700522874e+11 picosegundos com 6 simulações, atingindo um desempenho de 203,87 MCPS e 407,75 MIPS. Com estes resultados, a melhoria de

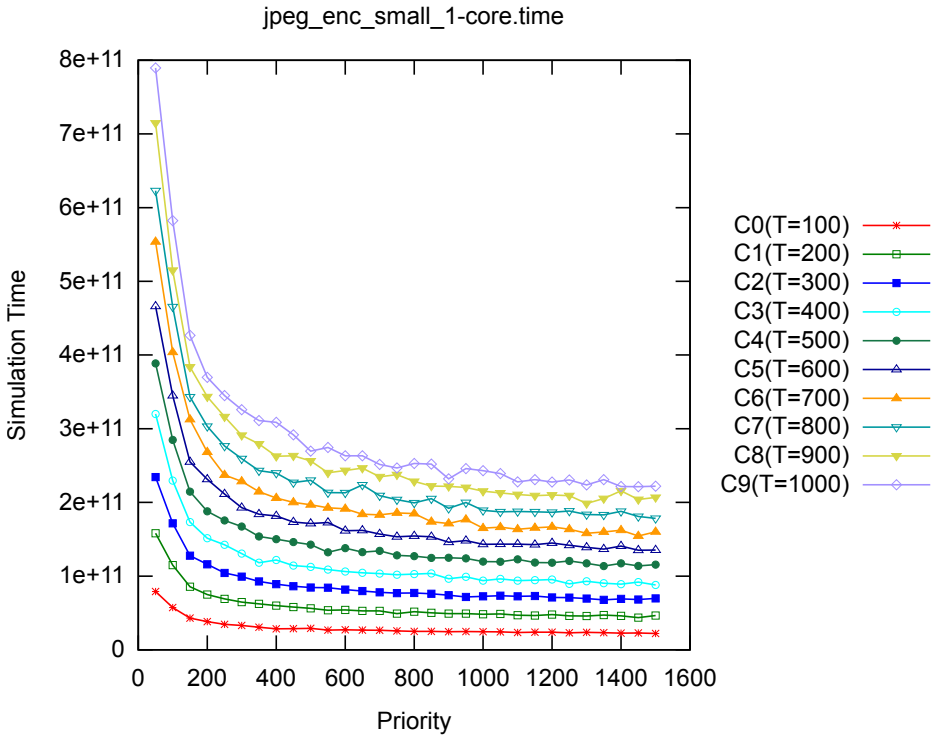


Figura 175: Tempo simulado de Codificador JPEG (Pequeno)

desempenho ficou em cerca de 156 e 593 vezes, respectivamente, com uma taxa de erro relativo ao ISS de aproximadamente 0,99% e desvio padrão de 2,406890194e+9 picosegundos (±2,09%).

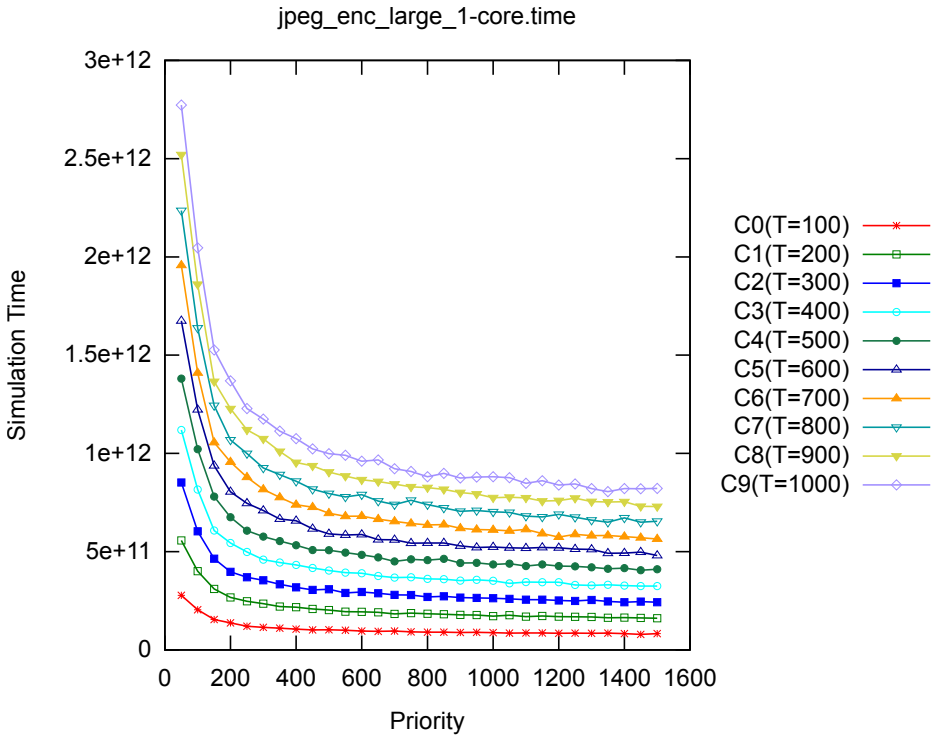


Figura 176: Tempo simulado de Codificador JPEG (Grande)

Na figura 176, são ilustradas as estimativas de tempo simulado para codificador JPEG com entrada de tamanho grande, usando a mesma parametrização dos exemplos anteriores e obtendo resultado bem similares. Na execução usando ISS foi obtido um tempo simulado de 4,30469731e+11 picosegundos, com um desempenho de 1,31 MCPS e 0,69 MIPS. Buscando equiparar o comportamento visto no ISS, os parâmetros de ajuste e prioridade são calibrados para os valores 542 e 1.033, respectivamente. Foi estimado um tempo simulado de 4,35605253743e+11 picosegundos em 5 simulações, com um desempenho de 215,36 MCPS e 430,72 MIPS. Com os resultados obtidos pelo modelo proposto, foi apurada uma melhoria de desempenho de cerca de 165 e 625 vezes, respectivamente, com um erro relativo a simulação com ISS de aproximadamente 1,19% e desvio padrão de 4,669803632e+9 picosegundos (±1,07%).

### B.2.3 Typeset

O pacote de aplicações MiBench possui um software chamada Typeset que captura as informações de um processador de HTML e faz sua composição para exibição. Esta aplicação é muito representativa, pois se trata de um elemento central de navegadores de Internet que estão presentes na grande maioria dos dispositivos embarcados modernos. Como vetores de teste são usados anúncios de lançamento de ferramentas (entrada pequena e grande) para sua composição e visualização em páginas.

Nas figuras 177 e 178, são exibidos os níveis de desempenho em MCPS e MIPS para a aplicação Typeset no modelo proposto. Com 10 parâmetros de ajuste (curvas C0 até C9), com valores de 100 até 1.000, com intervalos de tamanho 100, respectivamente, e priorização de 50 até 1.500, com intervalos de tamanho 50. Com um pico de desempenho de cerca de 600 MCPS e 1.200 MIPS, para a parametrização definida, foi obtida uma melhoria de desempenho de 472 e 1.714 vezes, respectivamente, quando comparado ao ISS com 1,27 MCPS e 0,70 MIPS.

Verificando o desempenho da aplicação Typeset para uma entrada de tamanho grande são obtidas as curvas visualizadas nas figuras 179 e 180. O comportamento obtido foi similar ao da entrada de tamanho pequeno, entretanto o pico de desempenho registrado foi de cerca de 600 MCPS e 1.200

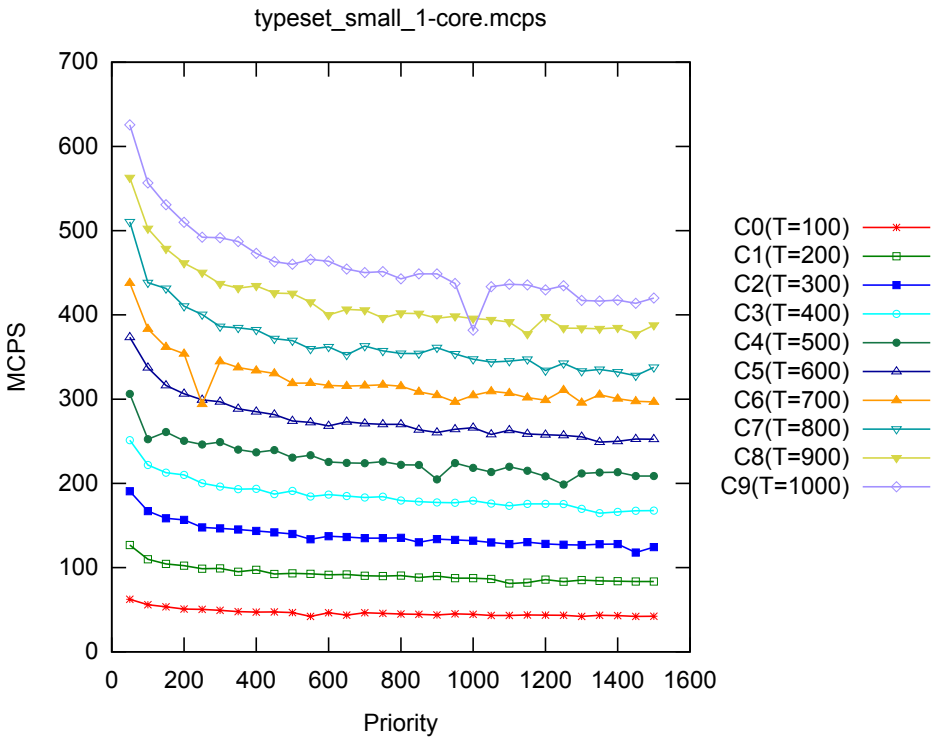


Figura 177: Desempenho em MCPS de Typeset (Pequeno)

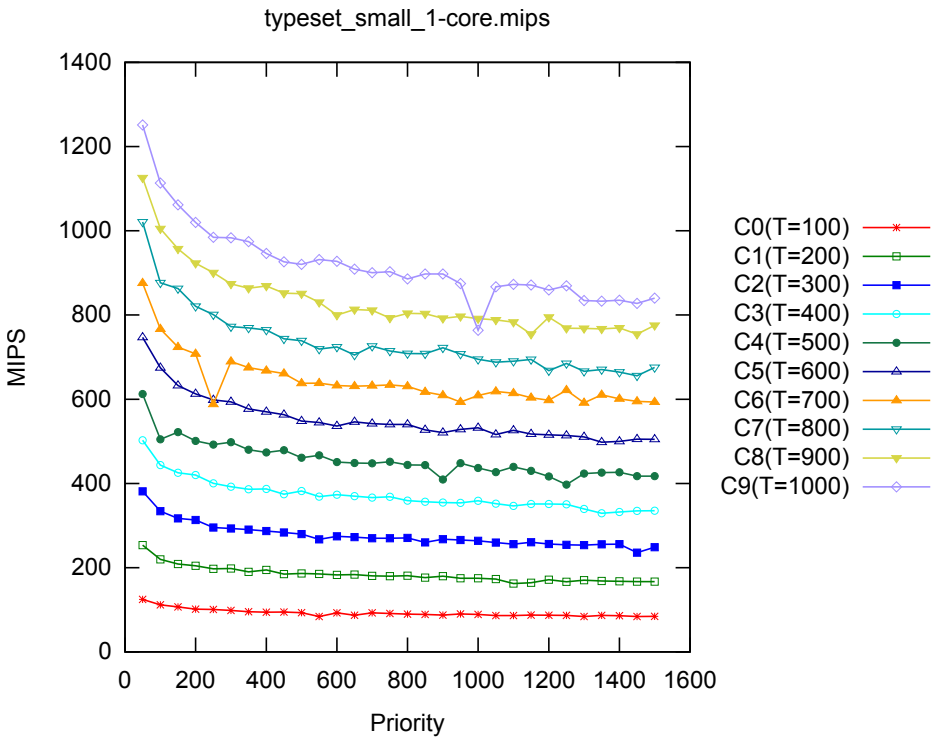


Figura 178: Desempenho em MIPS de Typeset (Pequeno)

MIPS, representando uma melhoria do desempenho em 465 e 1.690 vezes, respectivamente. O desempenho observado no ISS, para esta aplicação com entrada de tamanho grande foi de 1,29 MCPS e 0,71 MIPS.

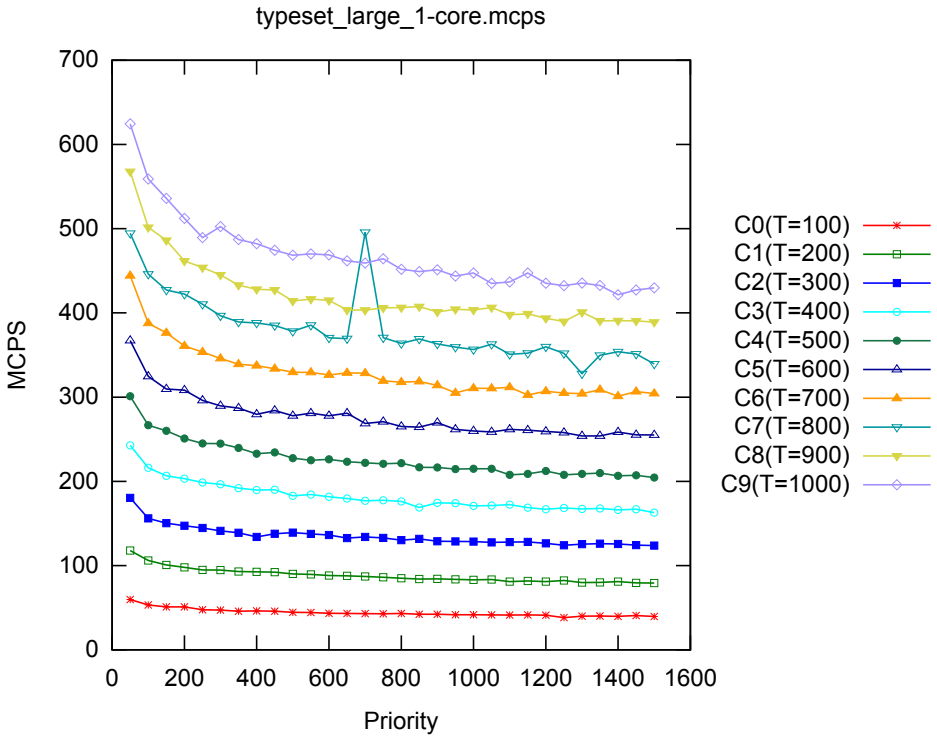


Figura 179: Desempenho em MCPS de Typeset (Grande)

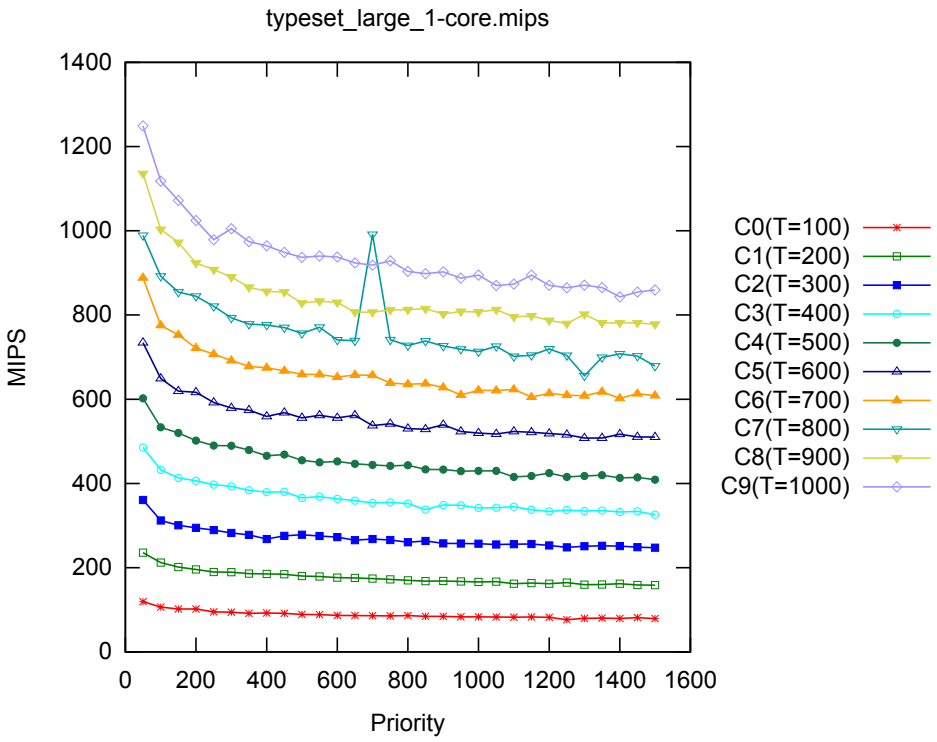


Figura 180: Desempenho em MIPS de Typeset (Grande)

Observando as estimativas de tempo simulado para uma entrada de tamanho pequeno, na figura 181, é possível verificar que o comportamento teórico previsto é comprovado e que os efeitos do não determinismo da simulação

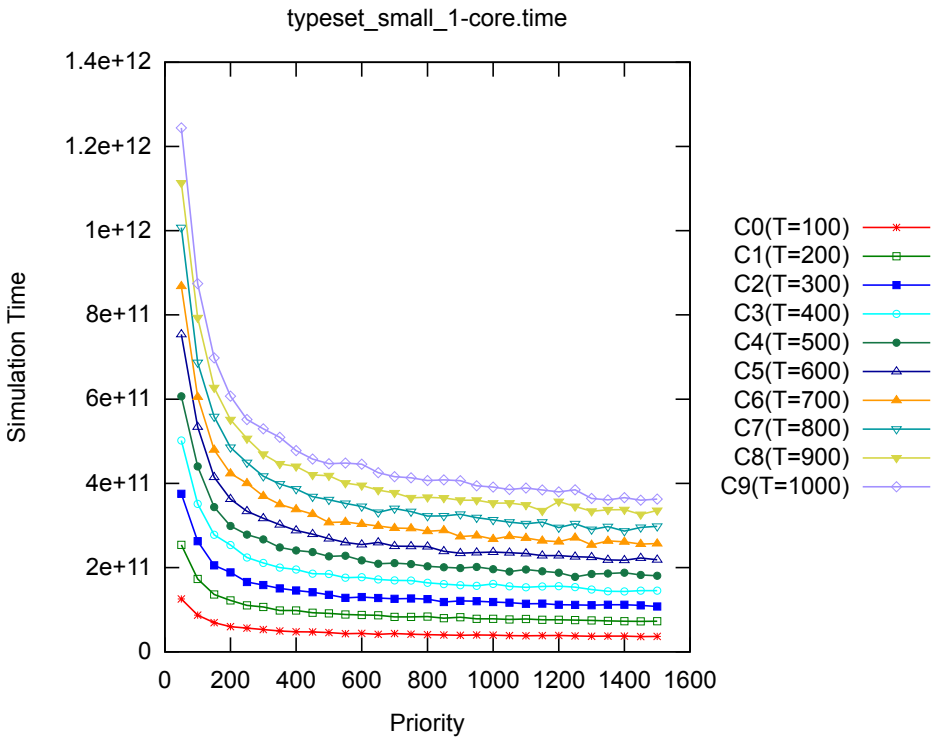


Figura 181: Tempo simulado de Typeset (Pequeno)

nativa são minimizados pelo emprego das técnicas propostas. No relatório de simulação do ISS foi obtido um tempo simulado de 3,60653274e+11 picosegundos, com desempenho de 1,27 MCPS e 0,70 MIPS. Para equiparar o comportamento do ISS, o modelo proposto recebe como parâmetros de ajuste e prioridade os valores 987 e 874, respectivamente, atingindo um tempo simulado de 3,65071088976e+11 picosegundos com 5 simulações. O desempenho do modelo proposto foi de 409,83 MCPS e 819,66 MIPS que representam uma melhoria de cerca de 323 e 1.176 vezes no desempenho de simulação, respectivamente, apresentando um erro relativo de aproximadamente 1,22% e desvio padrão de 4,378547727e+9 picosegundos (±1,19%) quando comparado ao ISS.

Com o aumento do tamanho da entrada, os efeitos do não determinismo são minimizados nas estimativas de tempo simulado, como pode ser visto na figura 182. O comportamento previsto continua válido e na simulação com ISS foi simulado um tempo de 2,179316027e+12 picosegundos, apresentando um desempenho de 1,29 MCPS e 0,71 MIPS. Mantendo o foco em abstrair o modelo ISS, precisam ser calibrados parâmetros de ajuste e de prioridade para aproximar o comportamento do ISS, maximizando o desempenho e minimizando o erro gerado. Para ajuste foi encontrado o valor de 1.172 e para

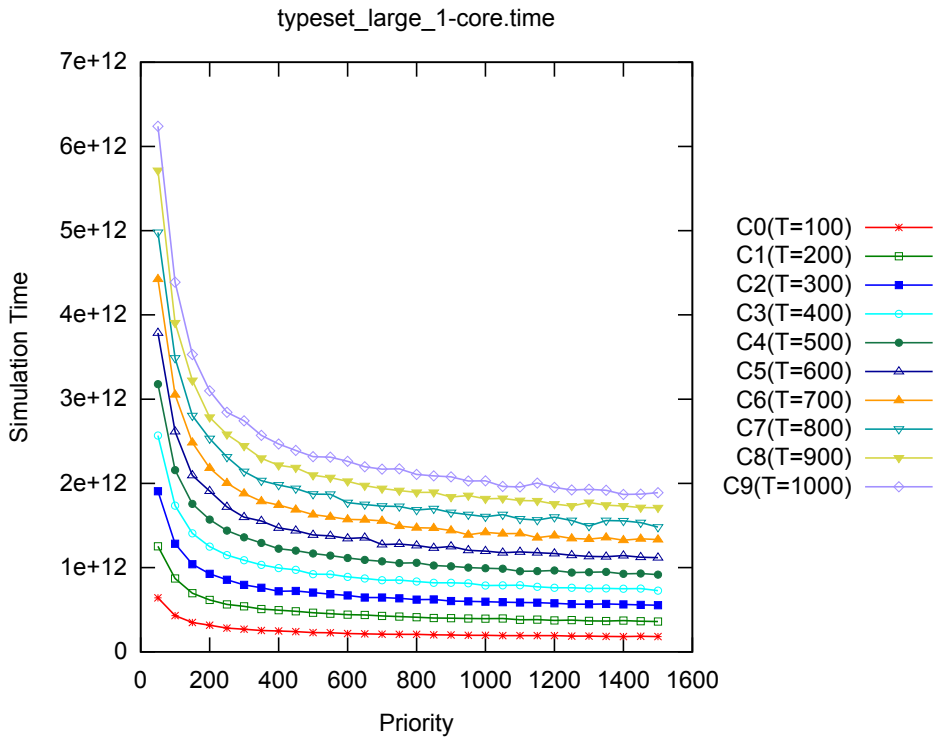


Figura 182: Tempo simulado de Typeset (Grande)

prioridade o valor de 986, obtendo uma estimativa de tempo simulado de 2,191444146738e+12 picosegundos em 7 simulações. Com desempenho de 487,27 MCPS e 974,54 MIPS, o modelo proposto abstrai o processamento do ISS com melhoria de desempenho de cerca de 377 e 1.382 vezes, respectivamente, com um erro relativo de 0,55% e desvio padrão de 2,1983398882e+10 picosegundos (±1,00%) com relação ao ISS.

## B.3 MiBench Network

Para avaliar as aplicações de rede, o pacote MiBench possui aplicações relevantes em diversos algoritmos de rede, seja para roteamento de pacotes ou para construção de estruturas de árvore para representação da rede.

### B.3.1 Dijkstra

O algoritmo de Dijkstra (63) que, a partir de uma fonte conhecida em grafo, é capaz de determinar a menor rota para qualquer outro ponto da rede. Por sua importância na área de redes, o conjunto de aplicações MiBench fornece

um software que implementa este algoritmo de roteamento, considerando uma representação em grafo da rede e todos os custos associados entre as conexões entre os nós. Para realização das análises são utilizadas entradas em arquivo com dois tamanhos (pequeno e grande), com todas as informações da rede e dos caminhos a serem processados.

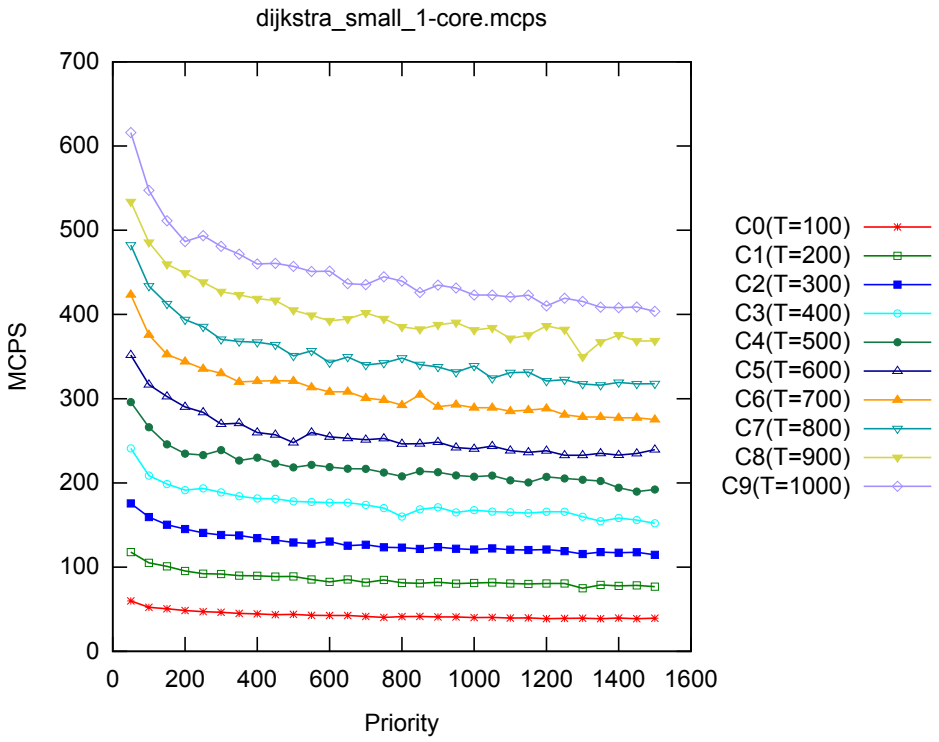


Figura 183: Desempenho em MCPS de Dijkstra (Pequeno)

Nas figuras 183 e 184, são exibidos os desempenhos em MCPS e MIPS para a implementação do algoritmo de Dijkstra utilizando uma entrada de tamanho pequeno. Podem ser observados 10 níveis de curva (C0 até C9), com valores de ajuste de 100 até 1.000, com passo de tamanho 100, e priorização de 50 até 1.500, com intervalos de tamanho 50. Foi atingido um pico de desempenho de 600 MCPS e 1.200 MIPS, considerando a parametrização definida, que representa uma melhoria de desempenho de cerca de 509 e 1.644 vezes, respectivamente, quando comparado ao modelo ISS que obteve 1,18 MCPS e 0,73 MIPS.

Aplicando entradas de tamanho grande, pode ser visto nas figuras 185 e 186 os gráficos de desempenho em MCPS e MIPS, respectivamente. Com a mesma parametrização definida e comportamento bastante similar foram obtidos picos de desempenho de cerca 600 MCPS e 1.200 MIPS. Estes resultados representam uma melhoria de desempenho de 496 e 1.622 vezes quando

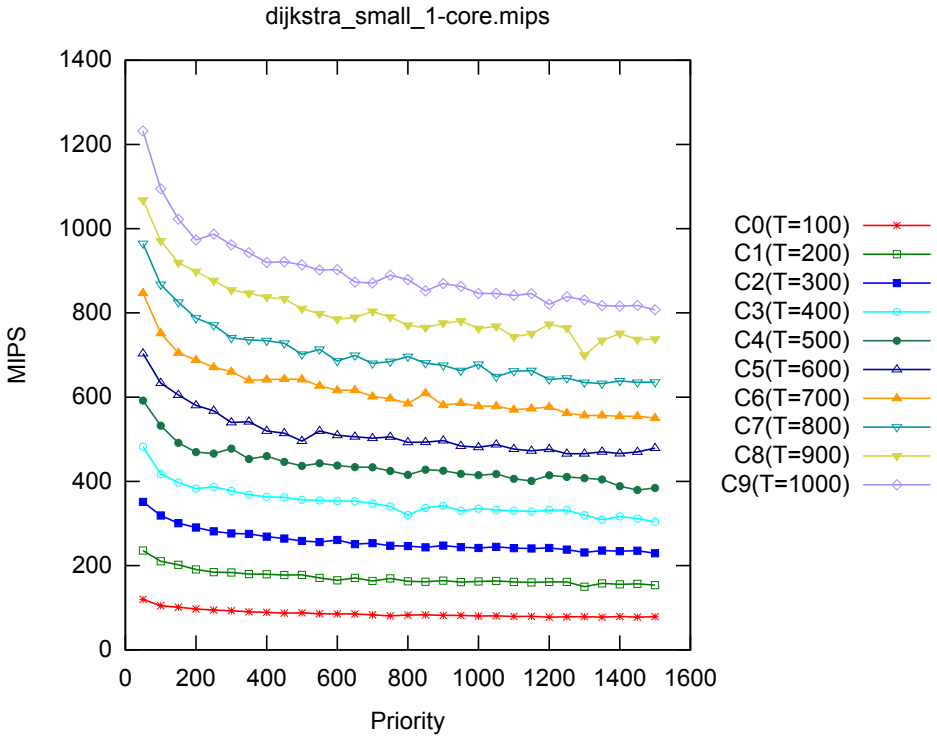


Figura 184: Desempenho em MIPS de Dijkstra (Pequeno)

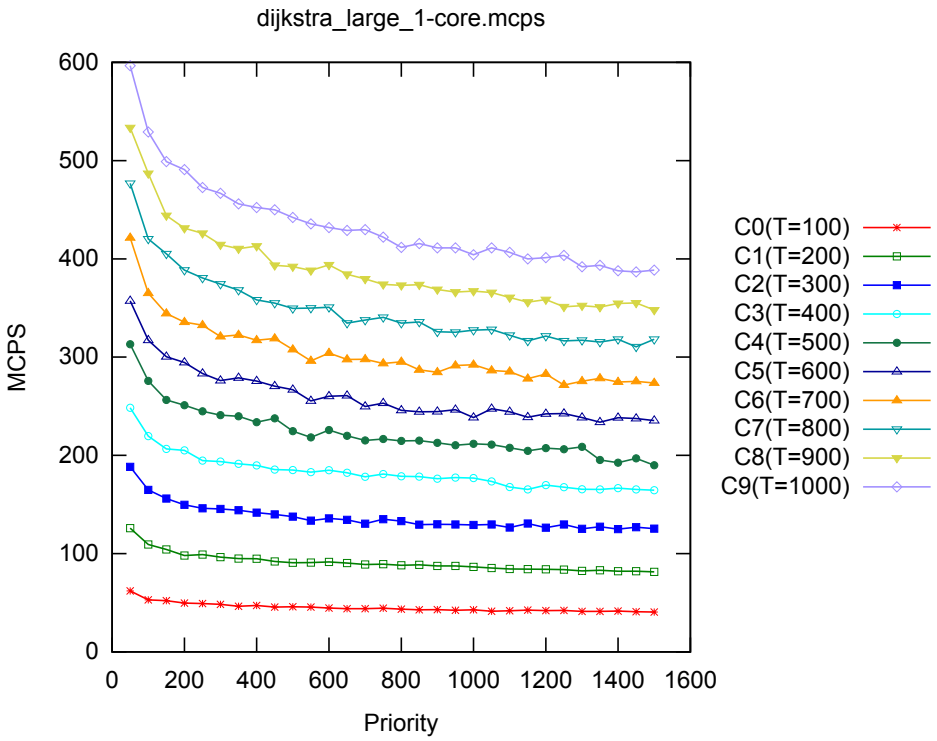


Figura 185: Desempenho em MCPS de Dijkstra (Grande)

comparado ao modelo ISS que obteve 1,21 MCPS e 0,74 MIPS.

Analisando as estimativas de tempo simulado na figura 187, pode ser observado mais uma vez o comportamento teórico previsto e pouco efeito do

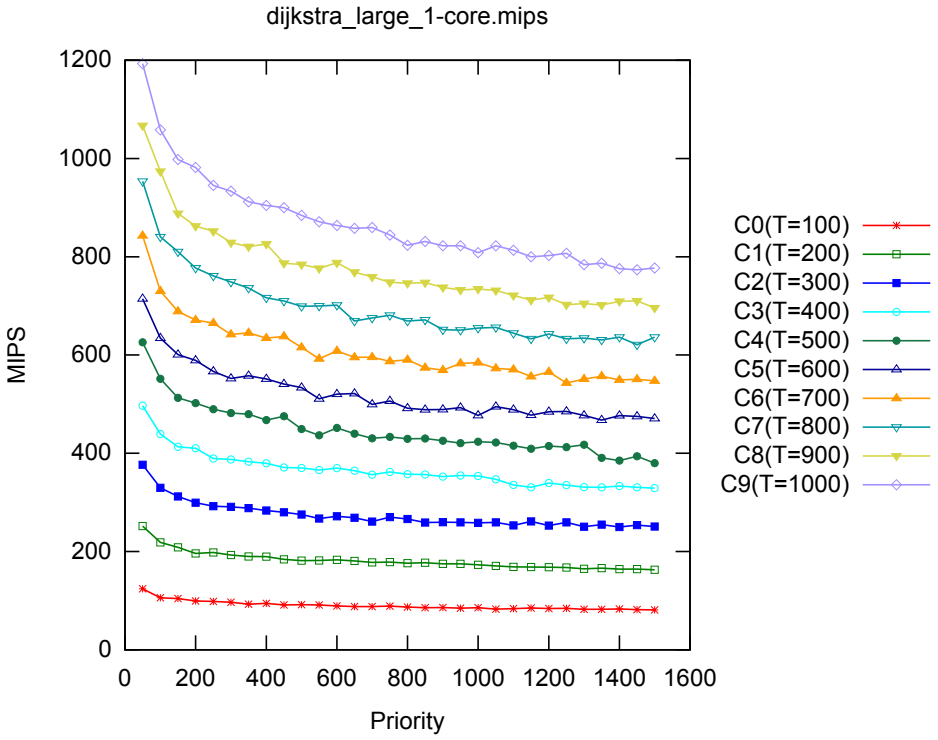


Figura 186: Desempenho em MIPS de Dijkstra (Grande)

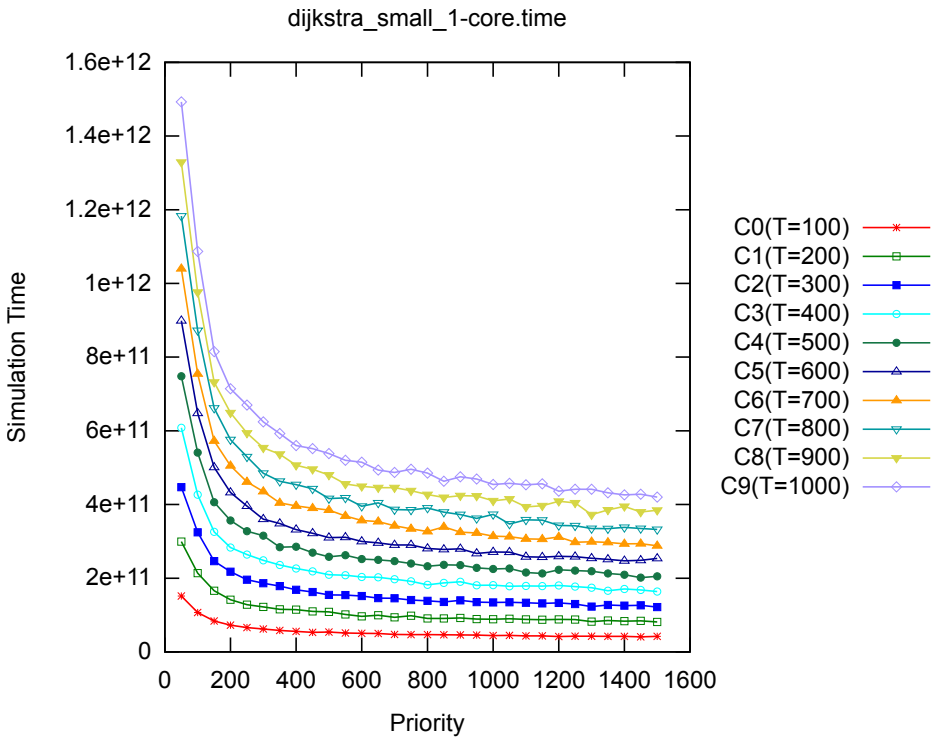


Figura 187: Tempo simulado de Dijkstra (Pequeno)

não determinismo do sistema nativo nas curvas, para uma entrada de tamanho pequeno. No resultado da simulação utilizando ISS foi obtido um tempo simulado de 2,15189634e+11 picosegundos e desempenho de 1,18 MCPS e

0,73 MIPS. Para que o modelo proposto apresente o mesmo comportamento do ISS, foram calibrados parâmetros de ajuste e de priorização com valores de 492 e 1.051, respectivamente. Com esta parametrização foi estimado um tempo simulado de 2,08873328571e+11 picosegundos em 10 simulações e desempenho de 193,80 MCPS e 387,61 MIPS que representam uma melhoria de 164 e 531 vezes, com um erro relativo ao ISS de cerca de 2,93% e desvio padrão de 3,799972595e+9 picosegundos (±1,81%).

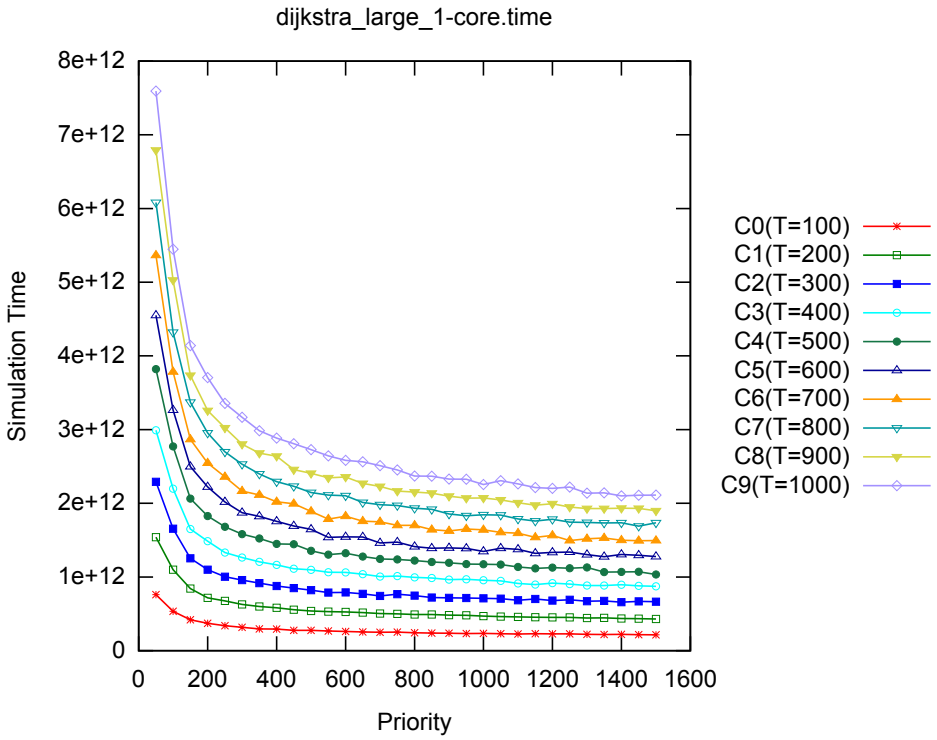


Figura 188: Tempo simulado de Dijkstra (Grande)

A figura 188 ilustra as estimativas de tempo simulado realizadas para uma entrada de tamanho grande, com comportamento similar a entrada pequena e as propriedades teóricas previstas no modelo. Na simulação com o modelo ISS foi obtido um tempo simulado de 1,036362295e+12 picosegundos com desempenho de 1,21 MCPS e 0,74 MIPS. Como o objetivo é abstrair o processamento do ISS mantendo o comportamento observado, são calibrados parâmetros de ajuste e de priorização com valores de 474 e 1.035, respectivamente, com estimativa de tempo simulado de 1,016553075718e+12 picosegundos em 4 simulações e desempenho de 189,29 MCPS e 378,58 MIPS. Estes resultados representam uma melhoria de desempenho de cerca de 157 e 511 vezes, com erro relativo ao ISS de aproximadamente 1,91% e desvio padrão de 6,44814068e+9 picosegundos (±0,63%).

### B.3.2 Patricia

Neste contexto, é fornecido no pacote de rede do MiBench uma aplicação chamada Patricia que faz uso intensivo destas estruturas de árvores para implementação de funcionalidades de rede. Sendo frequentemente utilizada em roteadores de rede, são utilizadas duas entradas de tamanho pequeno e grande que correspondem ao tráfego real de um servidor de internet de alta demanda durante o período de 2 horas.

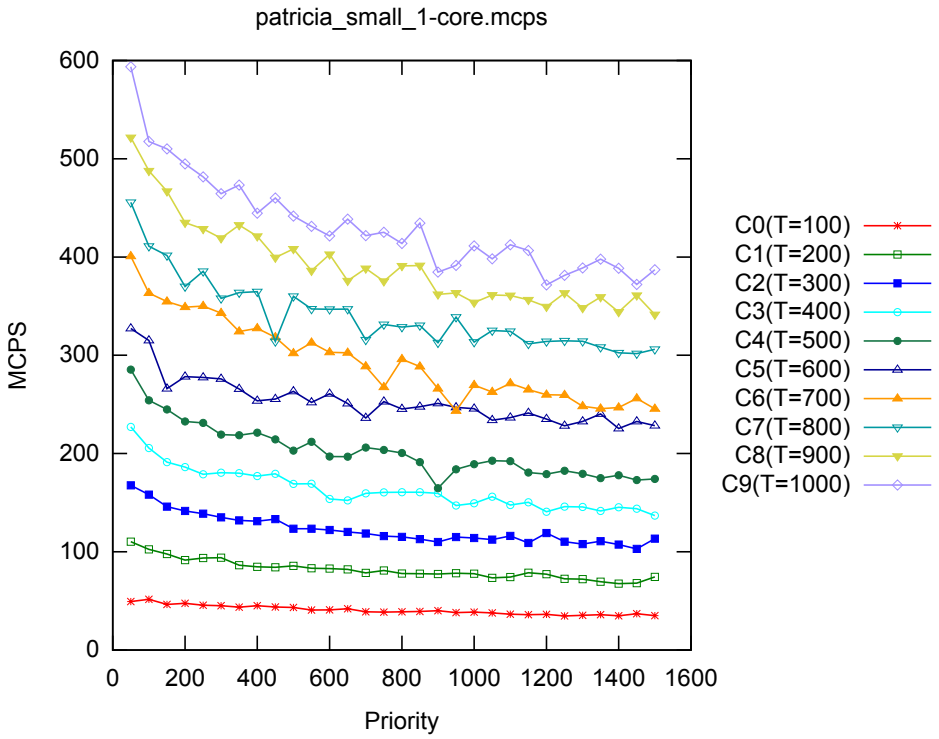


Figura 189: Desempenho em MCPS de Patricia (Pequeno)

Os desempenhos em MCPS e MIPS são exibidos nas figuras 189 e 190, respectivamente, para a aplicação Patricia com entrada de tamanho pequeno. É definida uma parametrização de 10 curvas de ajuste (C0 até C9), no intervalo de 100 até 1.000, com passos de tamanho 100, e de priorização de 50 até 1.500, com passos de tamanho 50. Nesta configuração foram atingidos picos de desempenho de 600 MCPS e 1.200 MIPS que representam uma melhoria de desempenho de 606 e 1.846 vezes quando comparado ao ISS que obteve 0,99 MCPS e 0,65 MIPS.

Nas figuras 191 e 192 são ilustrados os desempenhos em MCPS e MIPS, respectivamente, aplicando uma entrada de tamanho grande para o programa. A mesma parametrização definida na entrada de tamanho pequeno é utili-

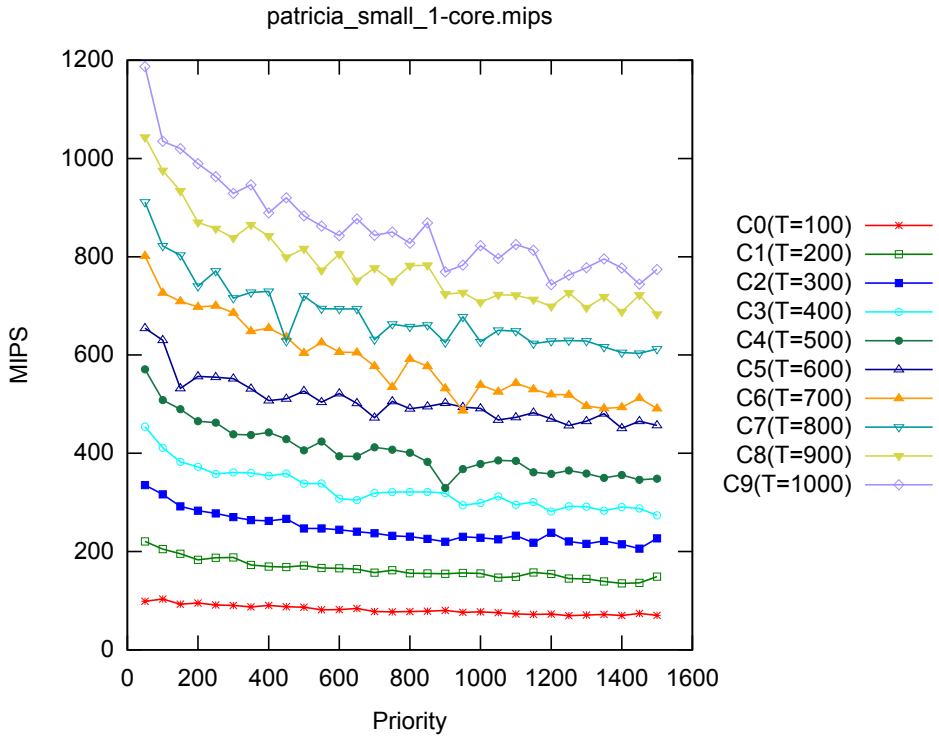


Figura 190: Desempenho em MIPS de Patricia (Pequeno)

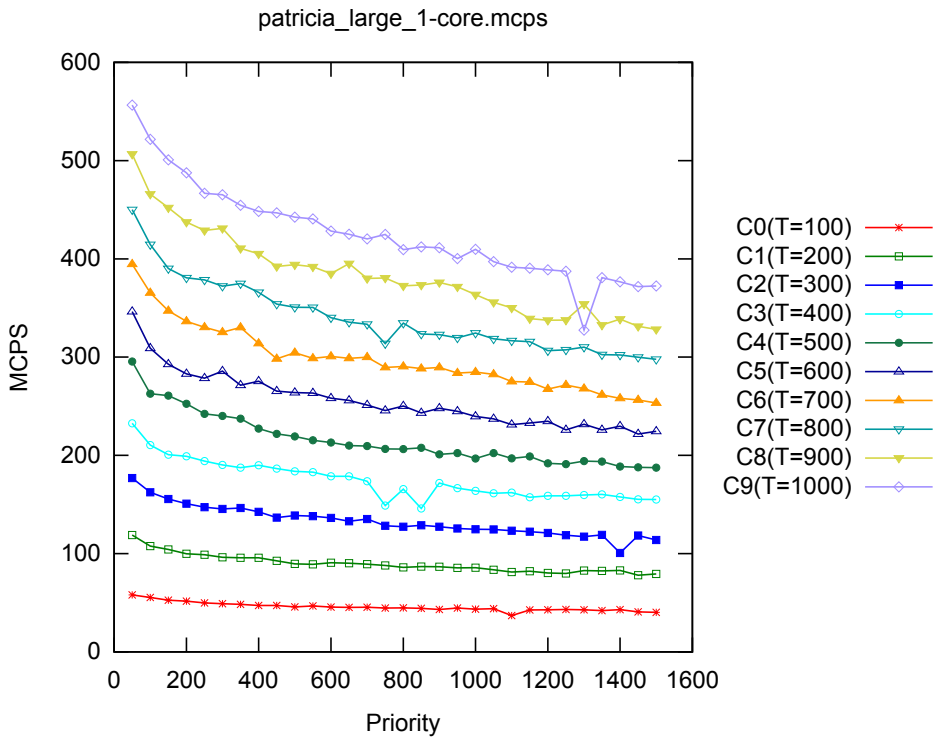


Figura 191: Desempenho em MCPS de Patricia (Grande)

zada e picos de desempenho de cerca de 550 MCPS e 1.100 MIPS são obtidos. Estes resultados representam uma melhoria de 561 e 1.692 vezes, respectivamente, quando comparada a simulação com ISS que obteve 0,98 MCPS e 0,65

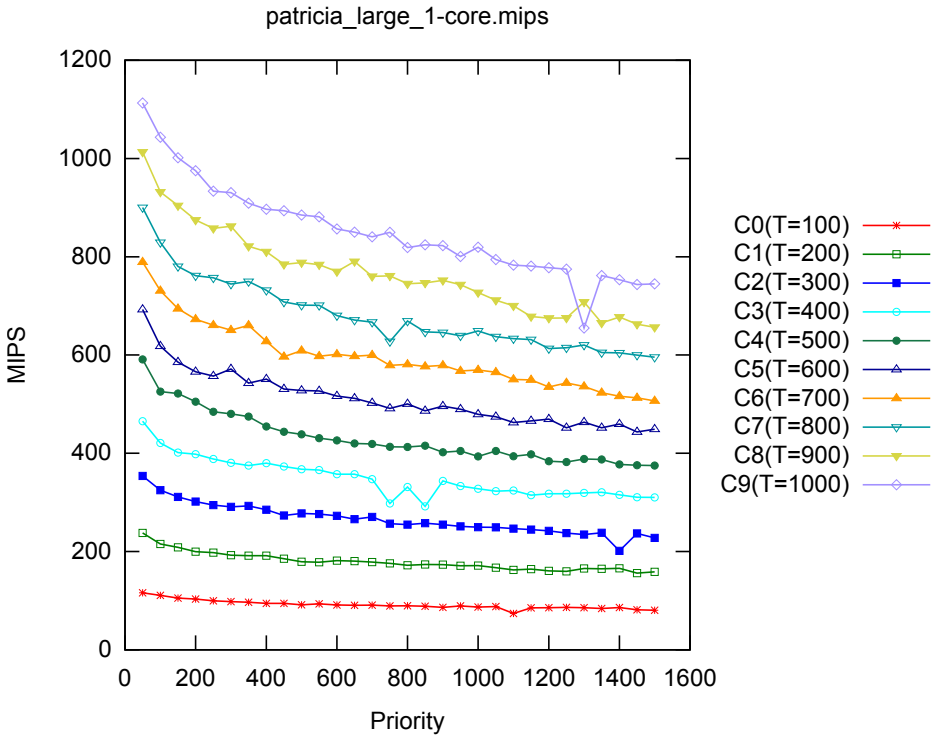


Figura 192: Desempenho em MIPS de Patricia (Grande)

MIPS.

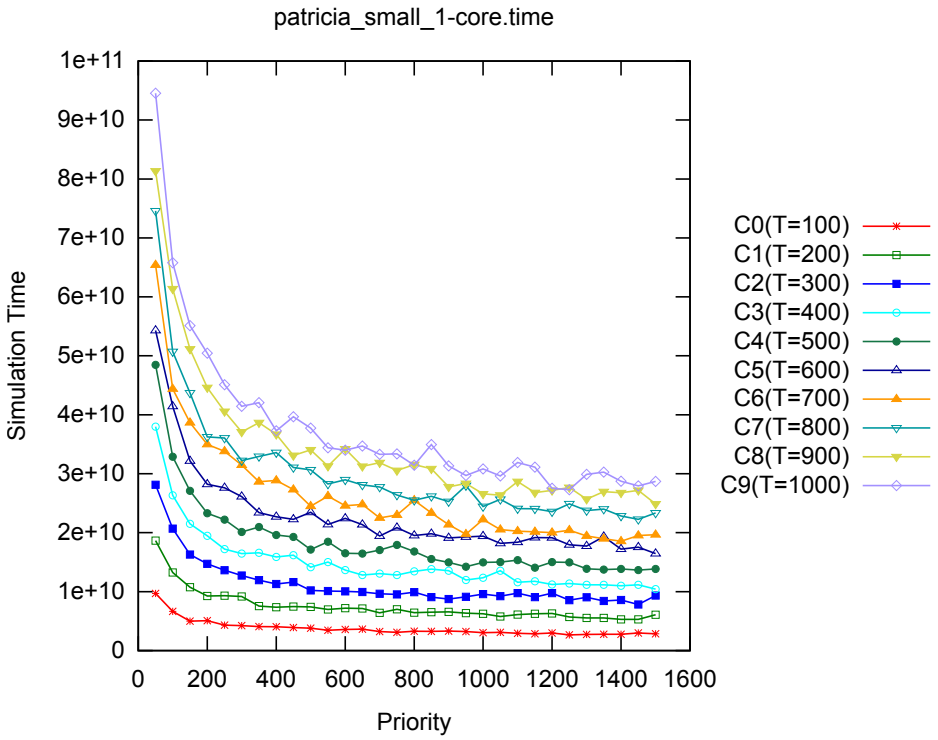


Figura 193: Tempo simulado de Patricia (Pequeno)

Concentrando a atenção nas estimativas de tempo simulado para uma entrada de tamanho pequeno, pode ser observado o resultado na figura 193.

Na simulação com ISS foi obtido um tempo simulado de 3,79899913e+11 picosegundos e desempenho de 0,99 MCPS e 0,65 MIPS. Para que o modelo proposto obtenha o mesmo comportamento observado no ISS, são calibrados parâmetros de ajuste e de priorização com valores de 13.083 e 1.159, respectivamente. O tempo simulado estimado desta configuração foi de 3,76348680611e+11 picosegundos, realizando 86 simulações, com desempenho de 4.922 MCPS e 9.845 MIPS, representando uma melhoria de desempenho de 4.953 e 15.078 vezes quando comparado ao ISS e com erro relativo de 0,93% e desvio padrão de 1,7835783724e+10 picosegundos (±4,73%).

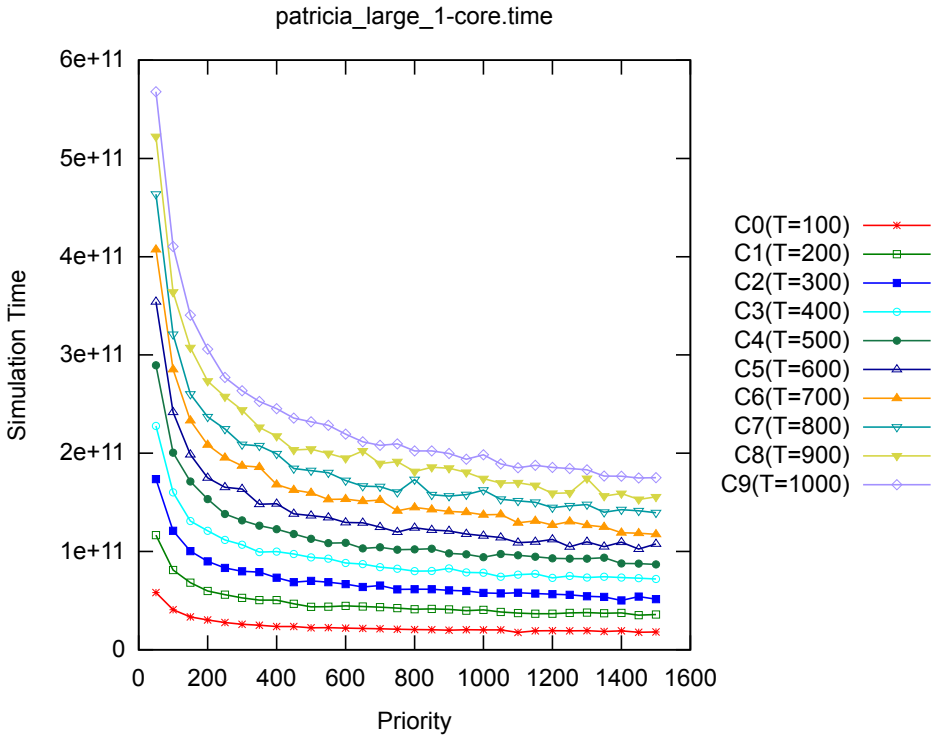


Figura 194: Tempo simulado de Patricia (Grande)

Utilizando agora uma entrada de tamanho grande, os efeitos do não determinismo são atenuados e a aplicação apresenta um comportamento mais regular, como pode ser visualizado na figura 194. Nos resultados obtidos com a simulação com ISS foi atingido um tempo simulado de 2,377838552e+12 picosegundos e desempenho de 0,98 MCPS e 0,65 MIPS. Para manter o mesmo comportamento observado no ISS, foram calibrados parâmetros de ajuste e de priorização com valores 12.839 e 1.005, respectivamente, que geraram uma estimativa de tempo simulado de 2,358554383088e+12 picosegundos com desempenho de 5.245 MCPS e 10.489 MIPS em 18 simulações. Estes resultados representam uma melhoria de 5.337 e 16.230 vezes, respectivamente, com

erro relativo ao ISS de cerca de 0,81% e desvio padrão de 4,4659699718e+10 picosegundos (±1,89%).

## B.4 MiBench Office

O conjunto de aplicações para escritório do MiBench possui aplicações para checagem ortográfica e busca de padrões no texto. Ambos os algoritmos são prevalentes em ferramentas de escritório e permitem uma avaliação realística da plataforma neste cenário de utilização.

### B.4.1 Ispell

Para ambientes Unix foi desenvolvido um sistema de checagem ortográfica batizado de Ispell (85), suportando a maioria das linguagens ocidentais. Por sua grande adoção em ambientes de código aberto, o pacote de Office do MiBench inclui a aplicação Ispell com duas entradas de tamanho pequeno e grande. Estas entradas são textos em língua inglesa que foram predefinidos com diversas características para que os algoritmos sejam bem exercitados e os resultados obtidos sejam mais consistentes.

Considerando a parametrização de ajuste de 10 curvas de valores 100 até 1.000, com intervalos de tamanho 100, e de priorização no intervalo de 50 até 1.500, com passos de tamanho 50, é feita geração dos gráficos de desempenho para entrada de tamanho pequeno em MCPS (figura 195) e em MIPS (figura 196) de Ispell. Nesta configuração são observados picos de desempenho de cerca de 600 MCPS e 1.200 MIPS que representam uma melhoria de aproximadamente 488 e 1.739 vezes, quando comparado ao desempenho obtido pelo ISS que foi de 1,23 MCPS e 0,69 MIPS.

Aplicando a mesma parametrização para ajuste e priorização definida anteriormente, a execução de Ispell para uma entrada de tamanho grande atinge picos de desempenho de cerca de 600 MCPS e 1.200 MIPS, como pode ser visualizado nas figuras 197 e 198, respectivamente. Este resultados trazem uma leve melhoria no desempenho obtido na entrada pequena e uma maior estabilidade do comportamento das curvas e consequentemente mais previsíveis. Este resultados geram uma melhoria de cerca de 484 e 1.714 vezes

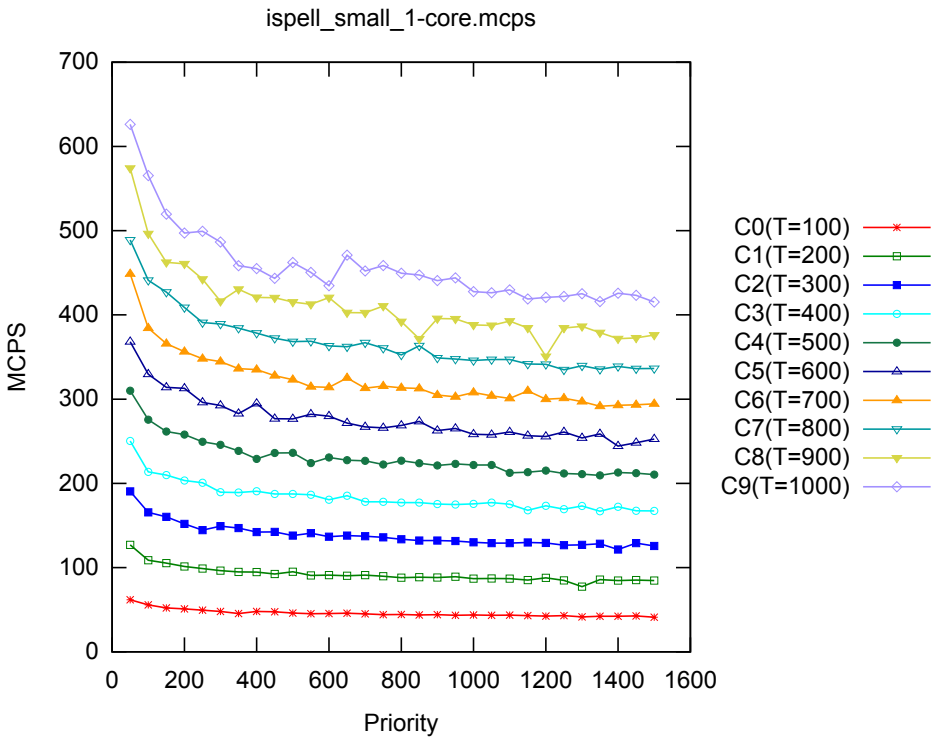


Figura 195: Desempenho em MCPS de Ispell (Pequeno)

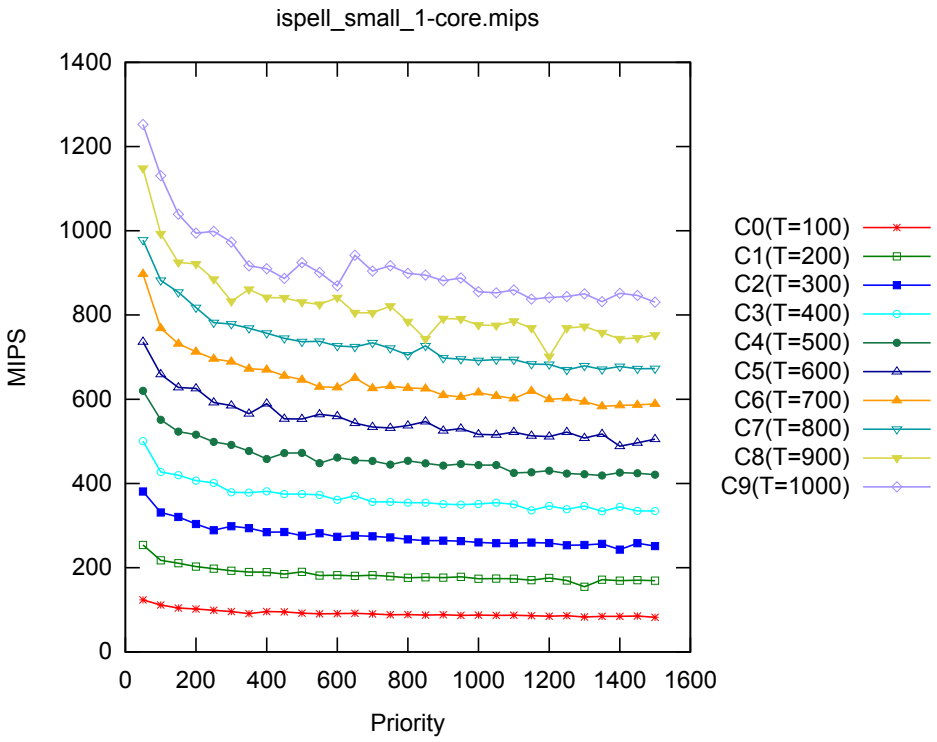


Figura 196: Desempenho em MIPS de Ispell (Pequeno)

no desempenho obtido pela simulação com ISS que obteve 1,24 MCPS e 0,70 MIPS.

Voltando a atenção para análise das estimativas de tempo realizadas,

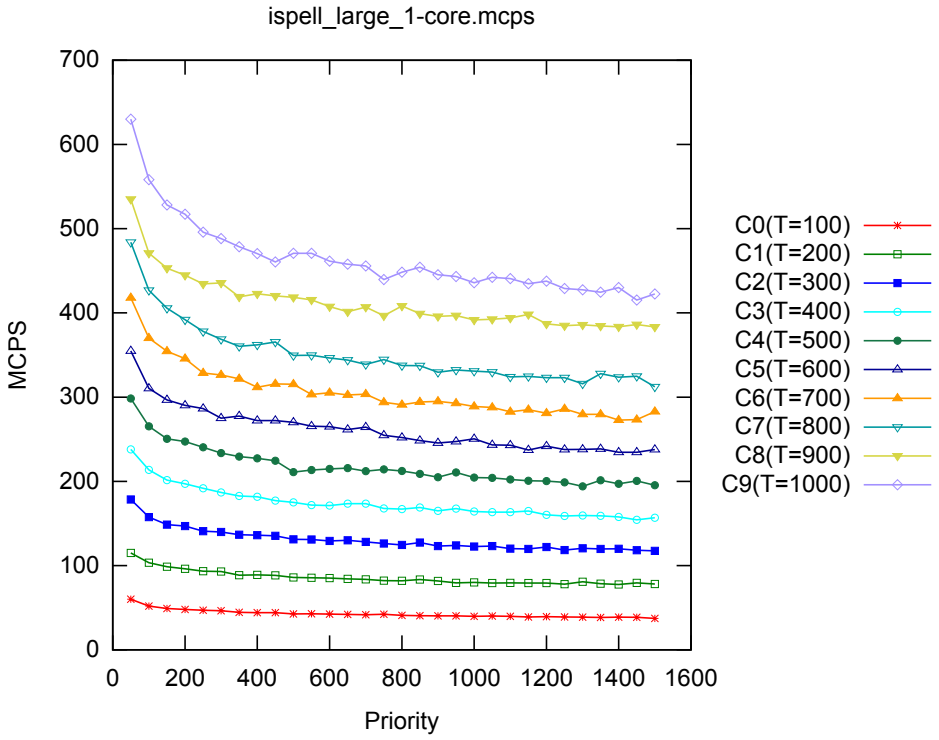


Figura 197: Desempenho em MCPS de Ispell (Grande)

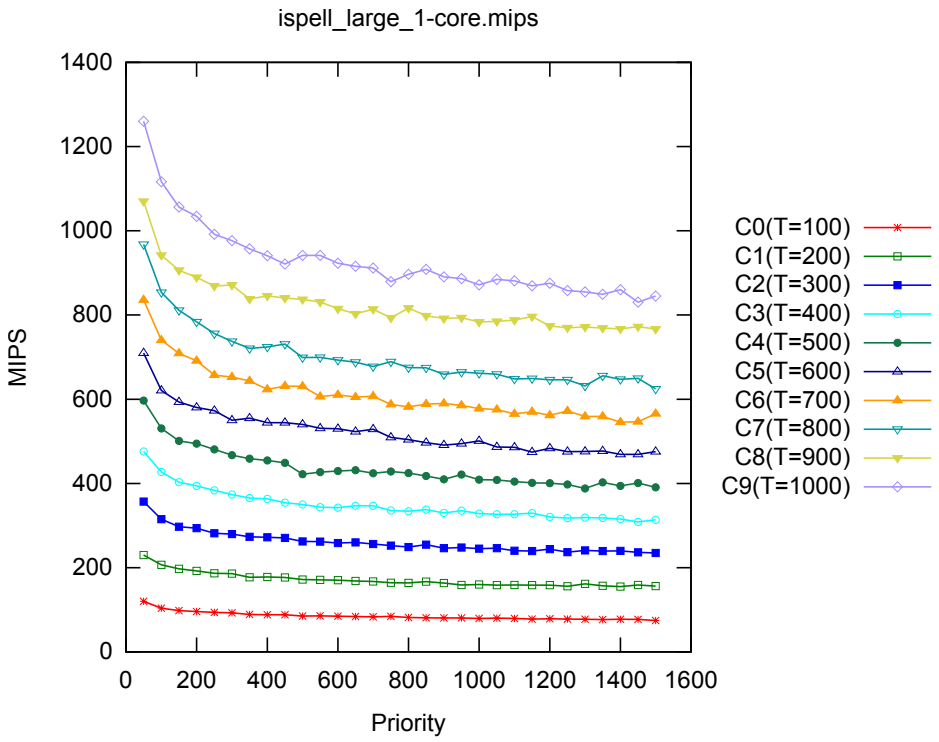


Figura 198: Desempenho em MIPS de Ispell (Grande)

pode ser observado na figura 199 o gráfico de tempo simulado para aplicação Ispell considerando uma entrada de tamanho pequeno. Na simulação com modelo ISS foi obtido um tempo simulado de 5,11433579e+11 picosegun-

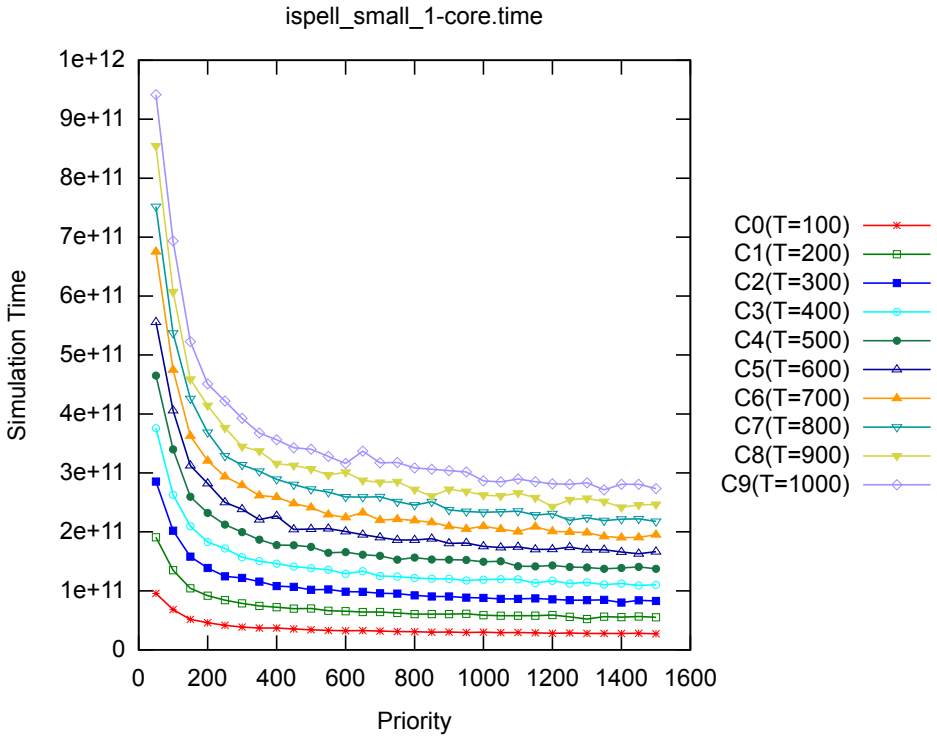


Figura 199: Tempo simulado de Ispell (Pequeno)

dos, com um desempenho de 1,23 MCPS e 0,69 MIPS. Como o objetivo é equiparar este comportamento obtido no ISS, são utilizados parâmetros de ajuste e de prioridade com valores de 1.889 e 1.091 que geram uma estimativa de tempo simulado de 5,04940524003e+11 picosegundos em 8 simulações. Além da estimativa de tempo, são obtidas taxas de desempenho de 756,35 MCPS e 1.513 MIPS que representam uma melhoria de desempenho de cerca de 616 e 2.188 vezes, respectivamente, com um erro de cerca de 1,26% e desvio padrão de 8,935230946e+9 picosegundos (±1,76%), quando comparado ao ISS.

Na figura 200 pode ser visto as estimativas de tempo simulado para entrada de tamanho grande de Ispell que foram obtidas utilizando a configuração de ajuste e priorização calibradas. A execução do Ispell com entrada de tamanho grande no ISS obteve um tempo simulado de 5,4327335228e+13 picosegundos, com desempenho de 1,24 MCPS e 0,70 MIPS. Para que um comportamento próximo ao obtido no ISS seja observado no modelo proposto, foram calibrados parâmetros de ajuste e de priorização com valores 1.398 e 1.044, respectivamente. Com esta configuração foi gerada uma estimativa de tempo simulado de 5,414356096076e+13 picosegundos em 4 simulações, com desempenhos associados de 575,97 MCPS e 1.152 MIPS, melhorias na veloci-

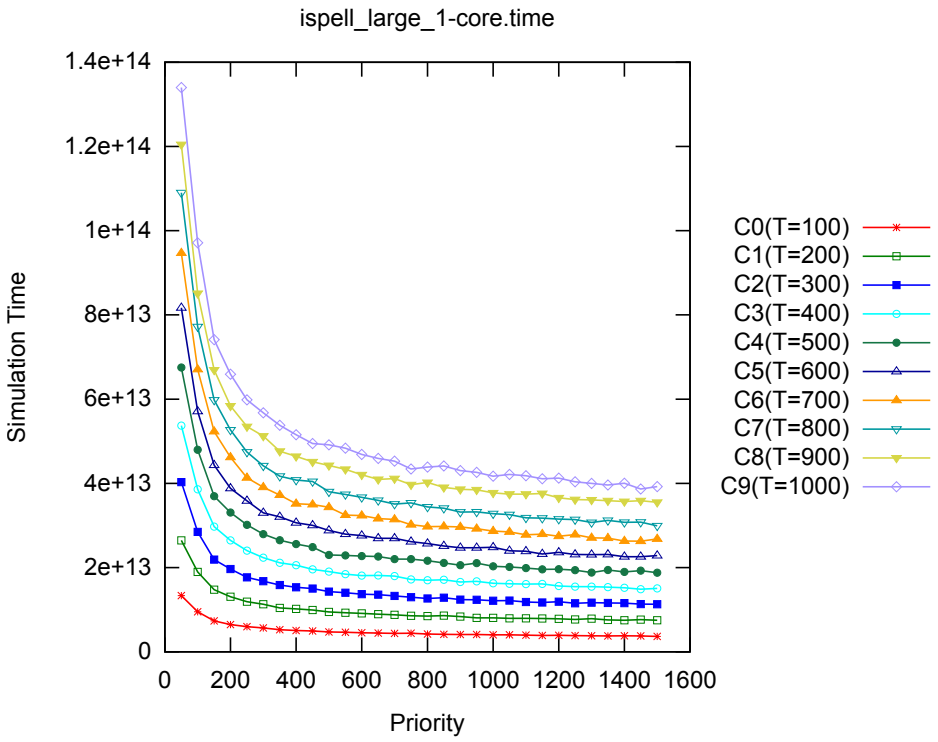


Figura 200: Tempo simulado de Ispell (Grande)

dade de simulação de 466 e 1.648 vezes, respectivamente, com um erro relativo de cerca de 0,33% e desvio padrão de 3,39415857756e+11 picosegundos (±0,62%), quando comparado ao modelo ISS.

### B.4.2 Stringsearch

Em diversas áreas da computação a busca de cadeias (string search) (63) é uma ferramenta essencial, seja para realizar o sequenciamento de uma cadeia de DNA ou realizar uma simples busca por palavras em um editor de textos. O pacote de aplicações Office do MiBench inclui a aplicação de Stringsearch com o objetivo de encontrar padrões em textos no formato ASCII, com tamanhos de entrada pequeno e grande.

Observando os gráficos de desempenho em MCPS e MIPS de Stringsearch, para entrada de tamanho pequeno, nas figuras 201 e 202, podem ser vistos picos de desempenho de cerca de 450 MCPS e 900 MIPS. Nesta configuração foram definidas 10 curvas de ajuste (C0 até C9), com valores de que vão de 100 até 1.000, com intervalos de tamanho 100, e valores de priorização de 50 até 1.500, com passos de tamanho 50. Comparando estes resultados com o modelo ISS que foram 0,36 MCPS e 0,21 MIPS, observa-se uma melhoria de

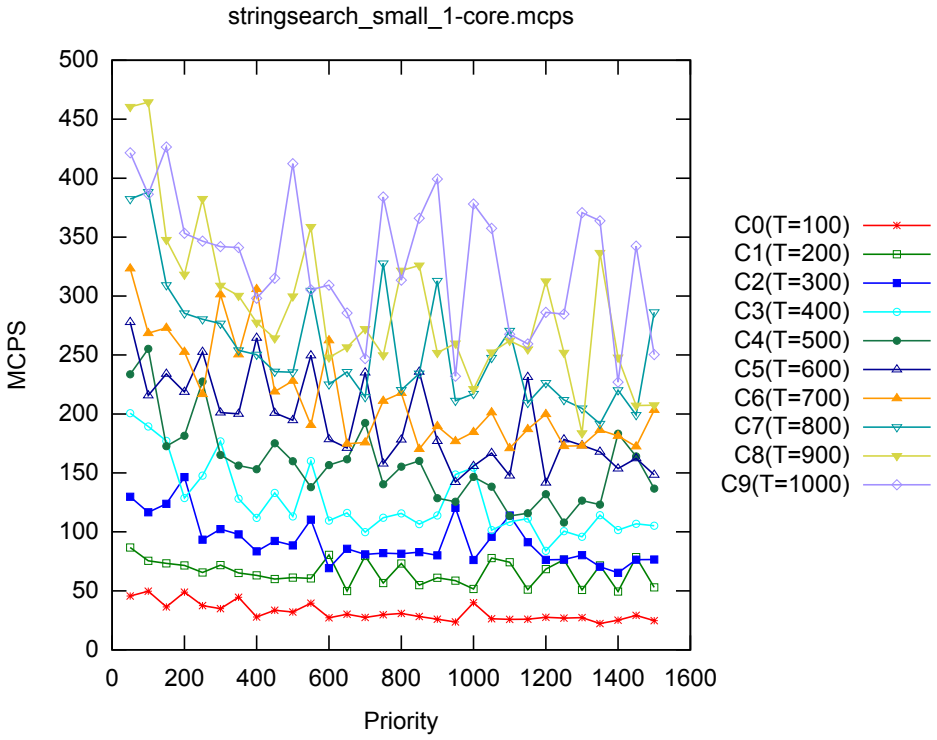


Figura 201: Desempenho em MCPS de Stringsearch (Pequeno)

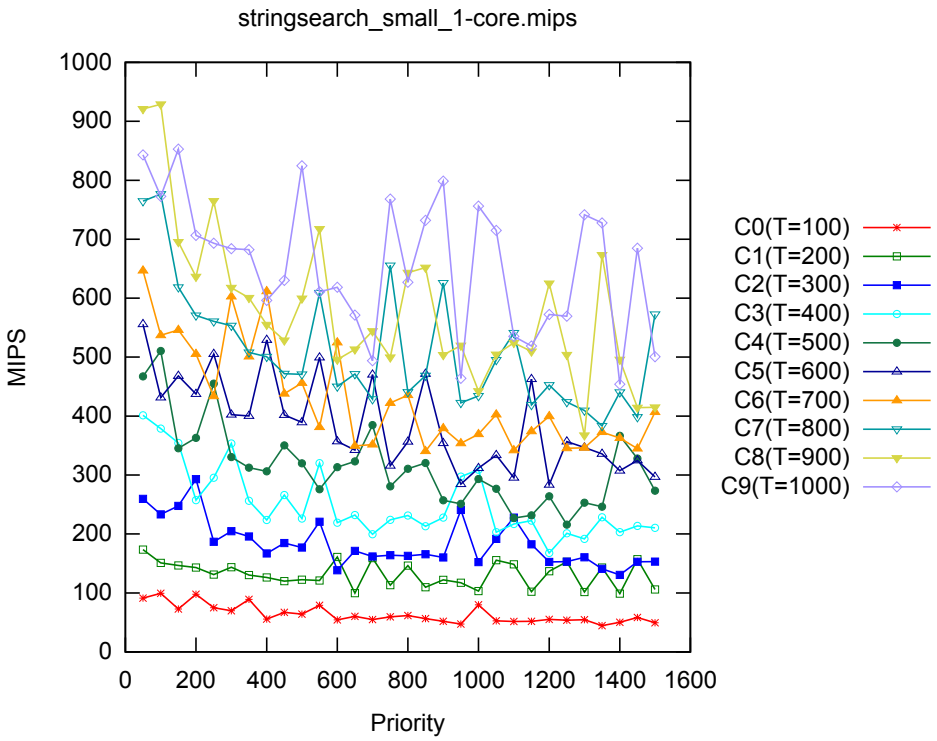


Figura 202: Desempenho em MIPS de Stringsearch (Pequeno)

desempenho de cerca de 1.250 e 4.286 vezes, respectivamente.

Adotando a mesma configuração de parametrização definida anteriormente, as curvas de desempenho da aplicação Stringsearch para entrada

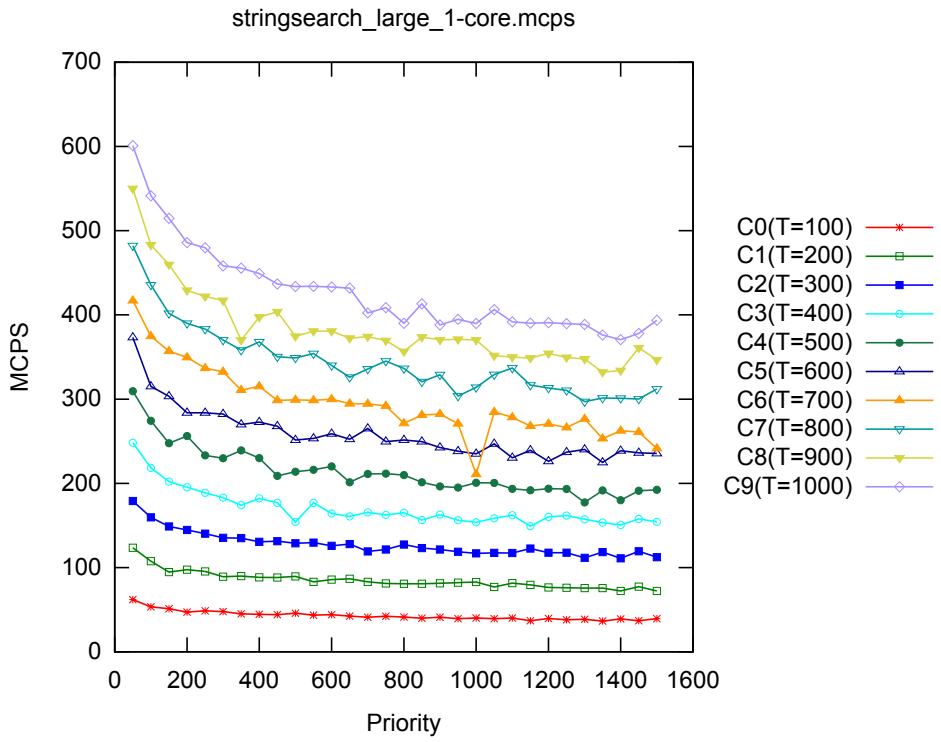


Figura 203: Desempenho em MCPS de Stringsearch (Grande)

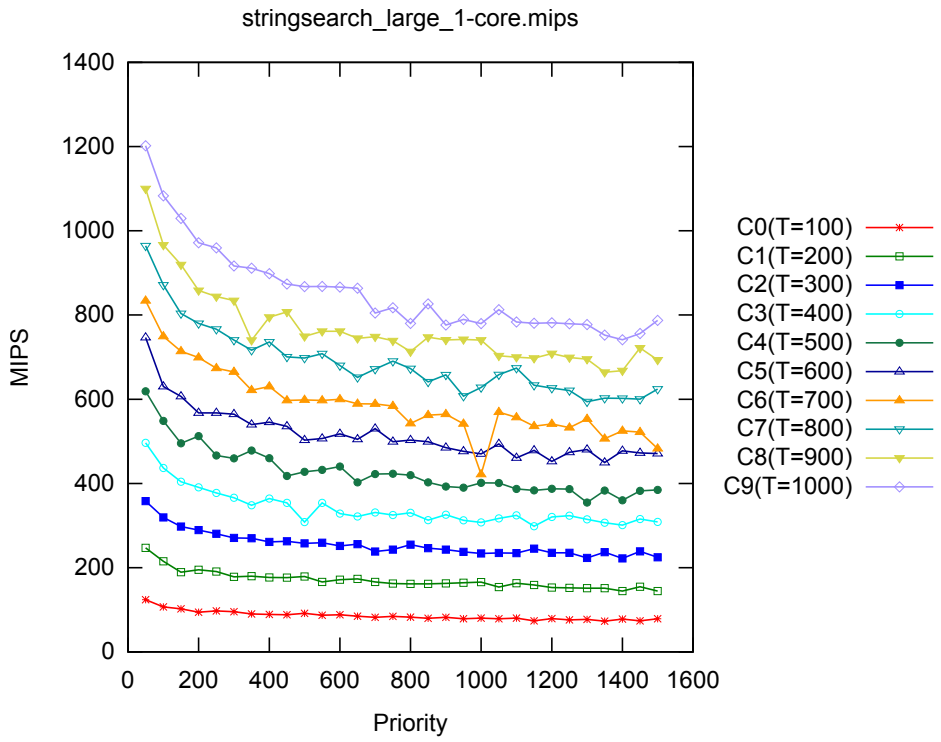


Figura 204: Desempenho em MIPS de Stringsearch (Grande)

de tamanho podem ser vistas nas figuras 203 e 204. Foram atingidos picos de cerca de 600 MCPS e 1.200 MIPS que significam um aumento de desempenho de 531 e 1.875 vezes, respectivamente, quando comparado ao modelo ISS

que obteve 1,13 MCPS e 0,64 MIPS.

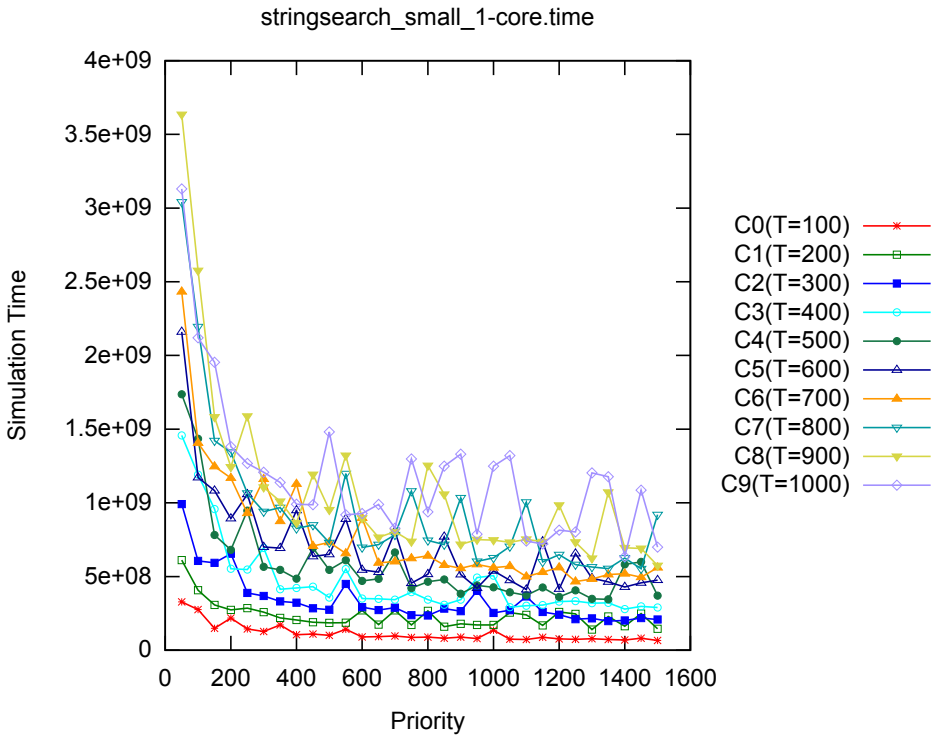


Figura 205: Tempo simulado de Stringsearch (Pequeno)

Focando agora na análise das estimativas de tempo simulado para entrada de tamanho pequeno, podem ser visualizadas na figura 205 as estimativas realizadas considerando a configuração de parâmetros de ajuste e priorização definidas anteriormente. Como a aplicação executa com entrada pequena, poucas iterações são realizadas, menos informações de tempo são coletadas e um comportamento com maior índice de erro é estimado. No modelo ISS foi atingido um tempo simulado de 6,96814e+8 picosegundos com um desempenho de 0,36 MCPS e 0,21 MIPS, para esta entrada de tamanho pequeno. Para equiparar este comportamento no modelo proposto, foi necessário utilizar parâmetros de ajuste e de priorização com valores de 667 e 820, respectivamente, para que fosse atingido um tempo simulado estimado de 6,33679692e+8 picosegundos em 1.192 simulações. Com relação aos resultados ISS, esta estimativa de tempo simulado apresenta um erro de 9,06% e desvio padrão de 1,20367602e+8 picosegundos (±18,99%), com desempenho de 190,62 MCPS e 381,24 MIPS que aumentam em 523 e 1.861 vezes a velocidade da simulação, respectivamente.

Realizando agora a análise das estimativas de tempo simulado para entrada de tamanho grande da aplicação Stringsearch, podem ser observadas

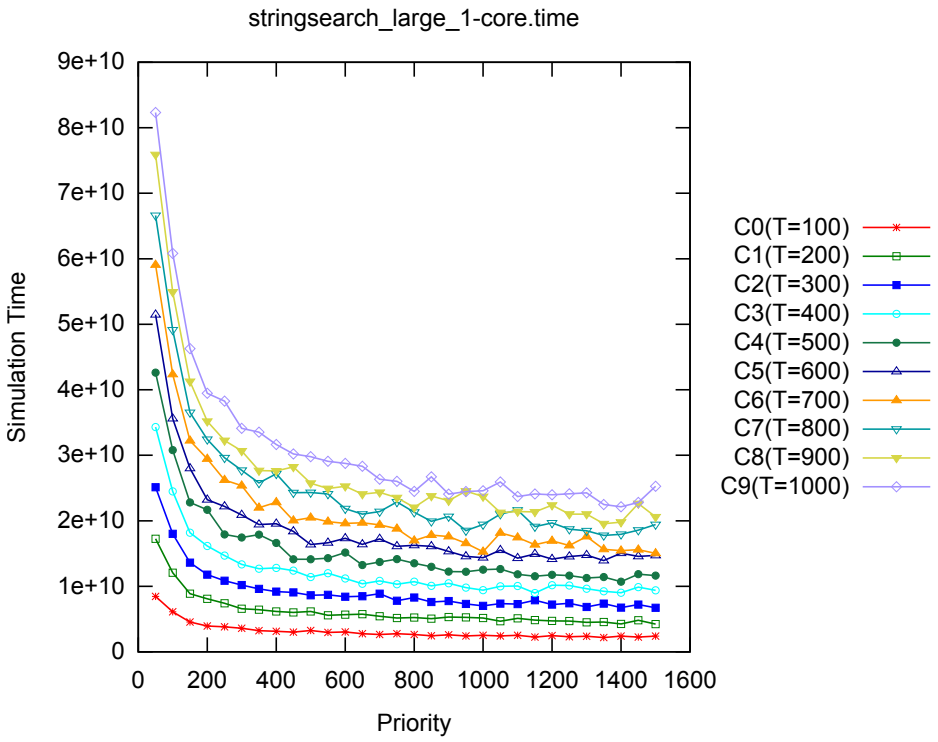


Figura 206: Tempo simulado de Stringsearch (Grande)

as curvas no gráfico da figura 206. Na simulação utilizando o ISS foi atingido um tempo simulado de 1,6743604e+10 picosegundos e desempenho de 1,13 MCPS e 0,64 MIPS. Como o objetivo é realizar estimativas sobre uma determinada plataforma alvo, medindo o erro e melhoria de desempenho gerados, são calibrados no modelo proposto os parâmetros de ajuste e de prioridade com valores de 671 e 838, respectivamente, que geram uma estimativa de tempo simulado de 1,6333767264e+10 picosegundos em 88 simulações, com erro relativo de 2,44% e desvio padrão de 7,96195935e+8 picosegundos (±4,87%). O desempenho obtido foi de 258,56 MCPS e 517,12 MIPS que melhoram cerca de 229 e 814 vezes o desempenho obtido pelo ISS.

## B.5 MiBench Security

As aplicações de segurança do pacote MiBench permitem avaliar o desempenho da plataforma em questão para diversos algoritmos de criptografia amplamente utilizados, desta maneira permitindo que uma avaliação de comportamento deste tipo de sistema possa ser feita.

### B.5.1 Blowfish Decoder

O algoritmo de decodificação Blowfish foi criado por Bruce Schneier em 1993 (86) e funciona através do conceito de chave privada e simétrica. O conjunto de aplicações MiBench em seu pacote de segurança implementa este algoritmo e faz sua execução com entradas de dados de tamanhos pequeno e grande. Assim, é esperado que o dado decodificado corresponda exatamente a mesma informação codificada pelo algoritmo, permitindo sua transmissão de forma sigilosa por um canal de comunicações aberto.

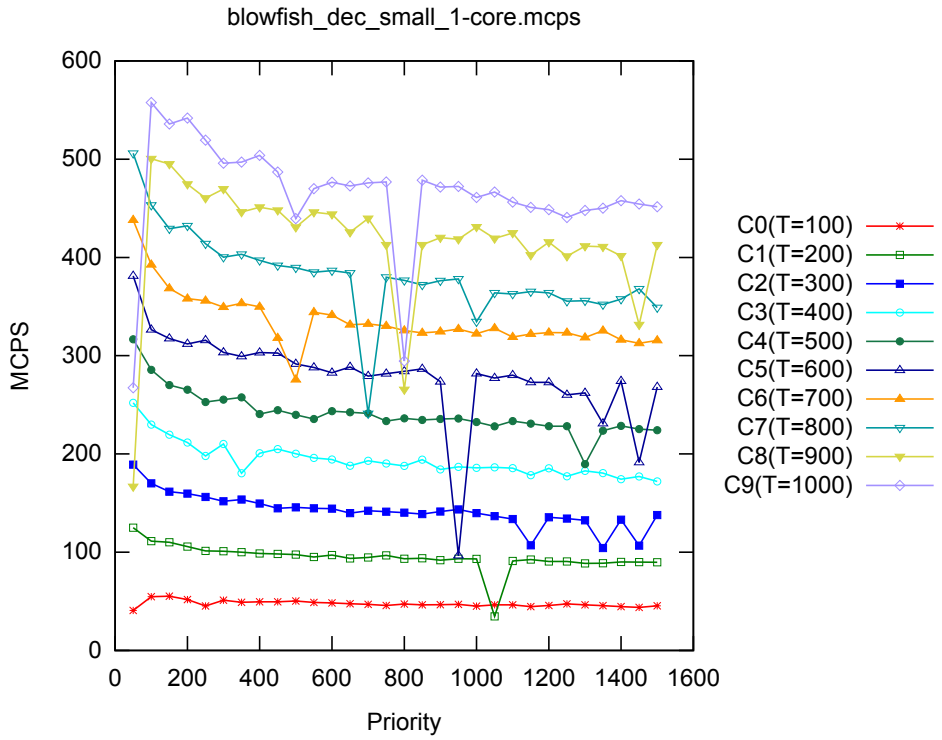


Figura 207: Desempenho em MCPS de Decodificador Blowfish (Pequeno)

Considerando a configuração de parâmetros de ajuste com 10 curvas (C0 até C9), com valores que partem de 100 até 1.000, com intervalos de tamanho 100, e de prioridade com intervalo de 50 até 1.500, com passos de tamanho 50, podem ser observados os gráficos de desempenho em MCPS (ver figura 207) e em MIPS (ver figura 208) para entrada de tamanho pequeno. Analisando as informações contidas nas curvas, pode-se observar picos de desempenho de cerca de 550 MCPS e 1.100 MIPS que representam uma melhoria de desempenho de 451 e 1.719 vezes, respectivamente, quando comparadas com a simulação em ISS que obteve 1,22 MCPS e 0,64 MIPS.

Adotando a mesma configuração de parâmetros de ajuste e de prioridade

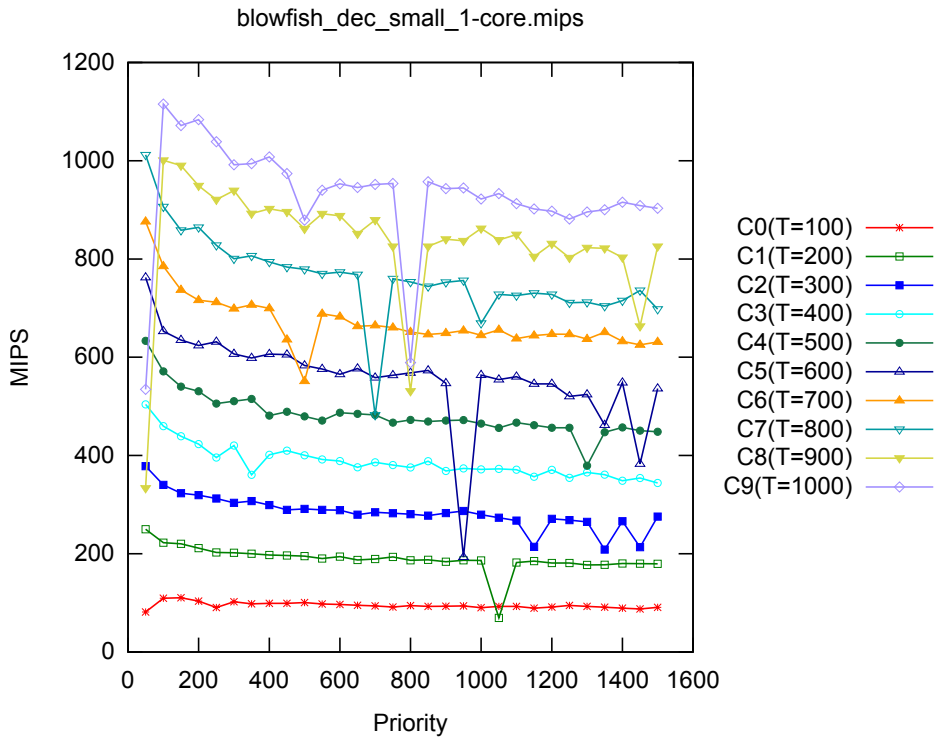


Figura 208: Desempenho em MIPS de Decodificador Blowfish (Pequeno)

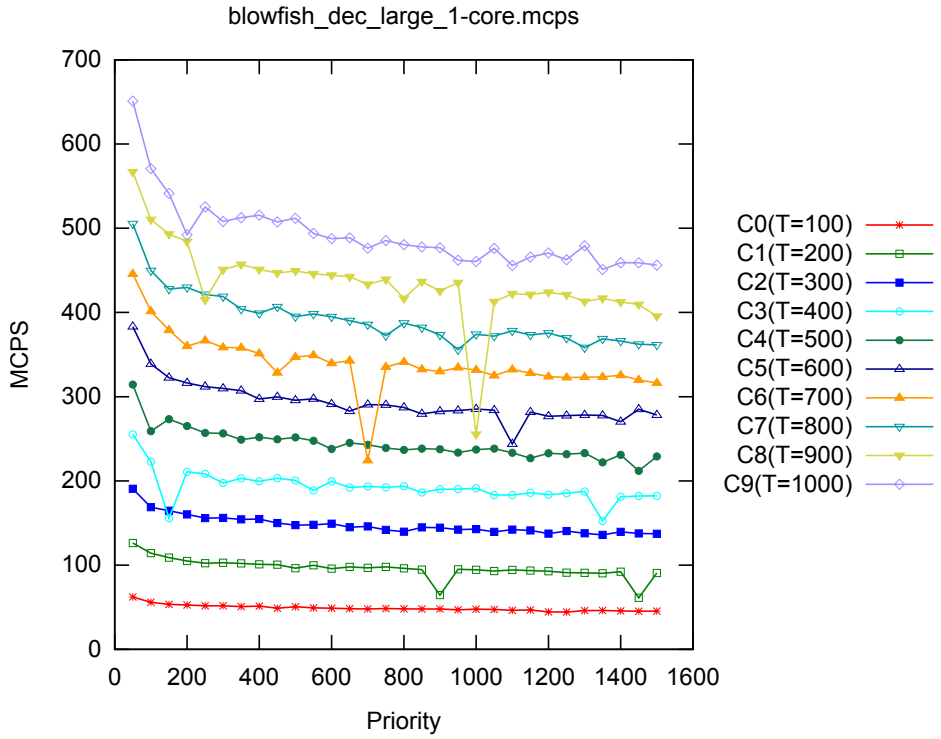


Figura 209: Desempenho em MCPS de Decodificador Blowfish (Grande)

descritas anteriormente, são gerados gráficos de desempenho em MCPS e MIPS para uma entrada de tamanho pequeno, como pode ser visto nas figuras 209 e 210, respectivamente. Analisando as curvas, podem ser vistos picos de

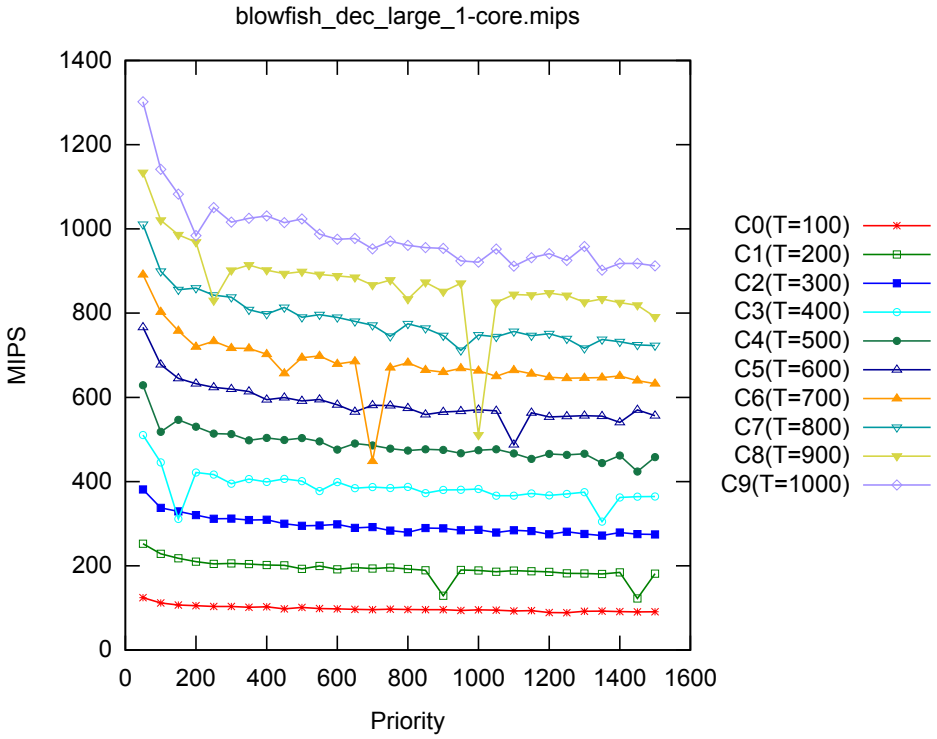


Figura 210: Desempenho em MIPS de Decodificador Blowfish (Grande)

desempenho de 650 MCPS e 1.100 MIPS que aumentam o desempenho em 508 e 1.642 vezes, respectivamente, quando comparado ao desempenho do ISS que foi de 1,28 MCPS e 0,67 MIPS.

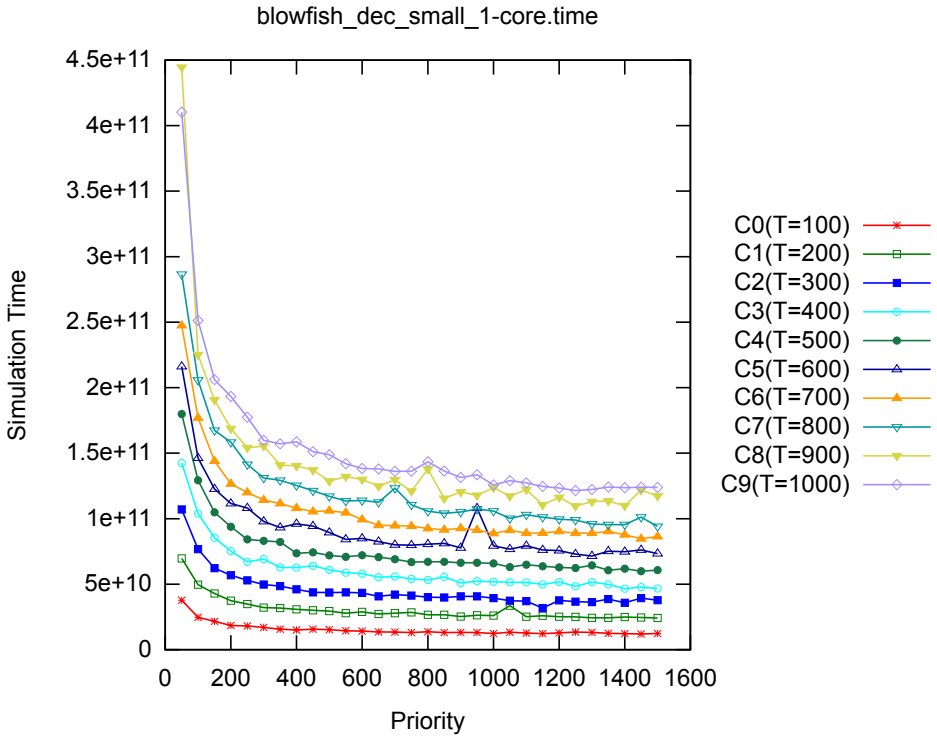


Figura 211: Tempo simulado de Decodificador Blowfish (Pequeno)

Focando-se nas estimativas de tempo simulado para entrada de tamanho pequeno, é feita a geração das curvas de tempo simulado estimado, seguindo a configuração de parametrização definida, como pode ser visualizado na figura 211. Na simulação no modelo ISS foi obtido um tempo simulado de 1,54354703e+11 picosegundos e desempenho de 1,22 MCPS e 0,64 MIPS. Como é desejado equiparar o comportamento observado no ISS e avaliar o erro da estimativa gerada, são calibrados parâmetros de ajuste e de prioridade com valores de 1.288 e 677, gerando uma estimativa de tempo simulado de 1,62777892266e+11 picosegundos em 12 simulações. Além da estimativa de tempo, foram obtidos 565,31 MCPS e 1.131 MIPS de desempenho que representam uma melhoria de 464 e 1.775 vezes sobre o ISS, com erro relativo de 5,45% e desvio padrão de 2,93393304e+9 picosegundos (±1,80%).

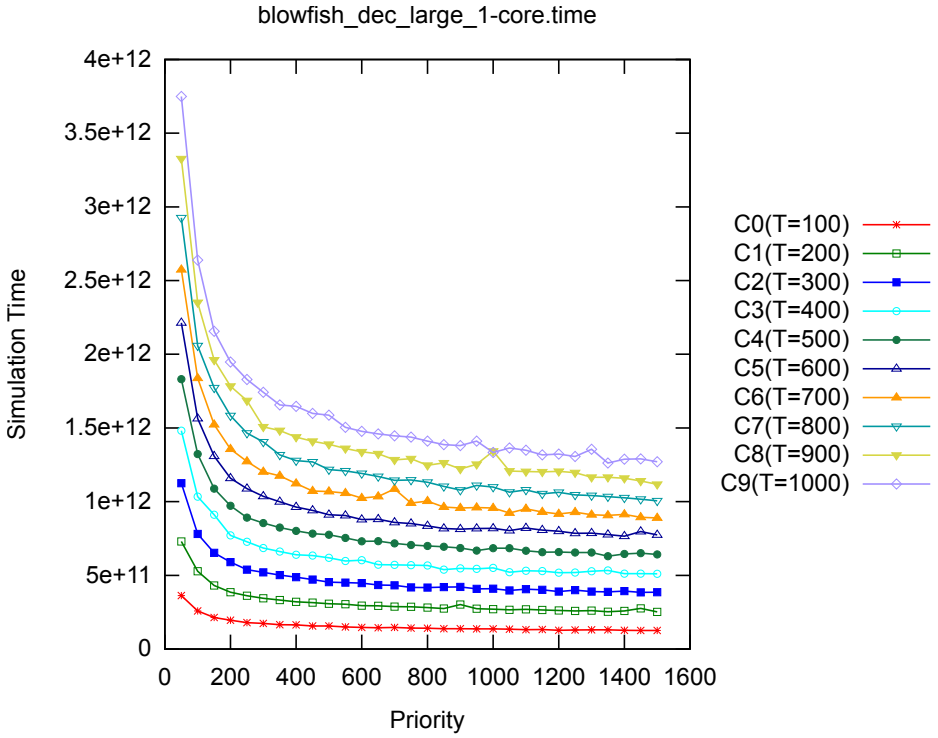


Figura 212: Tempo simulado de Decodificador Blowfish (Grande)

Na figura 212 é feita a exibição das estimativas de tempo simulado realizadas para entrada de tamanho grande, considerando a parametrização de ajuste e de prioridade definidas. Utilizando o modelo ISS foi atingido um tempo simulado de 1,600593972e+12 picosegundos e desempenho de simulação de 1,28 MCPS e 0,67 MIPS. Para igualar o comportamento do ISS foram calibrados os parâmetros de ajuste e de prioridade com valores de 1.285 e 1.433, respectivamente, que geraram uma estimativa de tempo simulado de

1,556003050227e+12 picosegundos em 5 simulações. Com erro relativo ao ISS de 2,78% e desvio padrão de 8,993614118e+9 picosegundos (±0,57%), esta simulação do modelo proposto obteve 554,71 MCPS e 1.109 MIPS de desempenho, tornando-o cerca de 435 e 1.665 vezes mais rápido que o ISS, respectivamente.

### B.5.2 Blowfish Encoder

Concebido em 1993 por Bruce Schneier (86), o algoritmo de codificação Blowfish é baseado em chave privada e simétrica para proteger um conjunto de dados que será enviado por canal sem segurança. Por suas características e relevância, o codificador do Blowfish foi incorporado ao pacote de segurança do MiBench, utilizando duas entradas de tamanho pequeno e grande que permitirão avaliar o comportamento sob diferentes demandas no sistema, recebendo um arquivo de entrada em texto plano e gerando um arquivo de saída criptografado protegido.

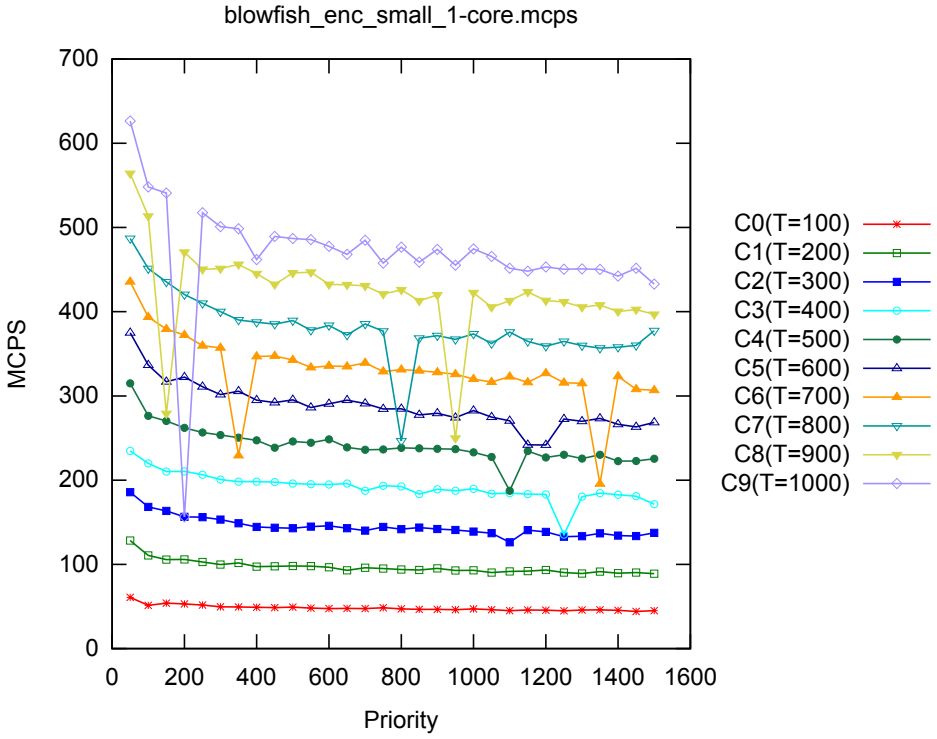


Figura 213: Desempenho em MCPS de Codificador Blowfish (Pequeno)

Nas figuras 213 e 214 são exibidas as curvas de desempenho em MCPS e MIPS, respectivamente, para a execução com entrada de tamanho pequeno. A parametrização considerada se baseou em 10 curvas de ajustes (C0 até C9),

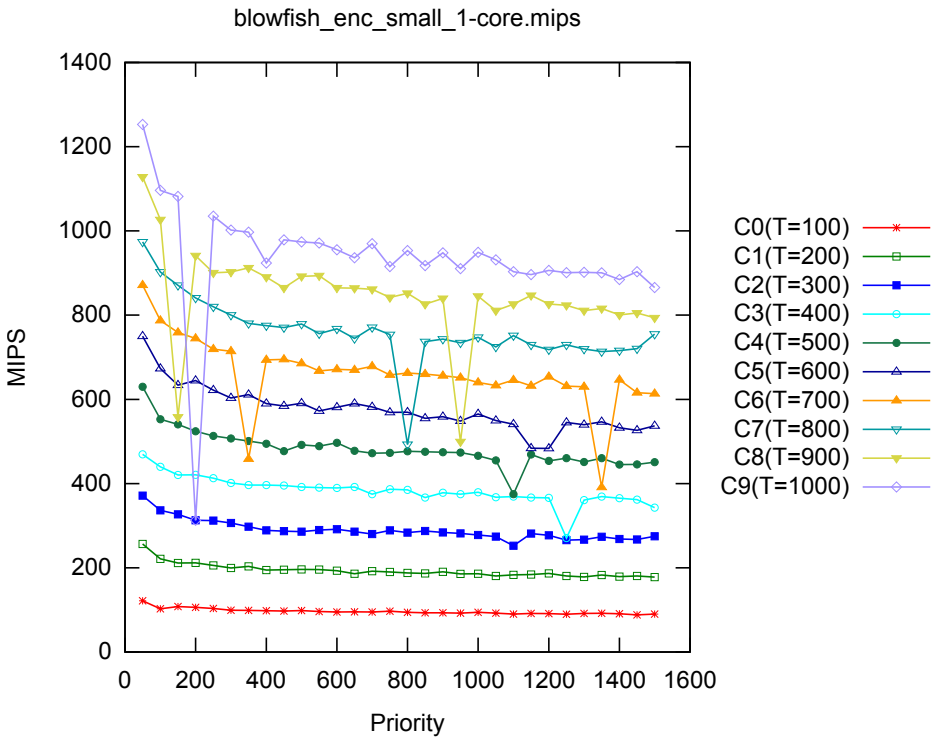


Figura 214: Desempenho em MIPS de Codificador Blowfish (Pequeno)

com valores de 100 até 1.000, com intervalos de tamanho 100, e de priorização de 50 até 1.500, com passos de tamanho 50. Observando-se as curvas, é percebido picos de desempenho de cerca de 600 MCPS e 1.200 MIPS que significam uma melhoria de desempenho de cerca de 500 e 1.905 vezes sob o ISS que obteve 1,20 MCPS e 0,63 MIPS.

Utilizando entrada de tamanho grande, foram gerados os gráficos vistos nas figuras 215 e 216, respectivamente, adotando a mesma parametrização já descrita anteriormente. Foram observados picos de desempenho de aproximadamente 650 MCPS e 1.300 MIPS que representam um aumento de cerca de 529 e 2.031 vezes com relação ao ISS que obteve 1,23 MCPS e 0,64 MIPS.

Analisando agora as estimativas de tempo simulado geradas para entrada de tamanho pequeno, observa-se na figura 217 o comportamento teórico previsto com um pequeno índice de erro nas estimativas geradas. Utilizando o modelo ISS para realizar a simulação da plataforma foi obtido um tempo simulado de 1,52211988e+11 picosegundos e desempenho de 1,20 MCPS e 0,63 MIPS. Para equiparar este comportamento observado no ISS e avaliar o erro da estimativa, foram utilizados parâmetros de ajuste e de prioridade com valores de 1.270 e 1.040, respectivamente. Esta configuração gerou uma estimativa de tempo simulado de 1,51438877715e+11 picosegundos em 377 simulações,

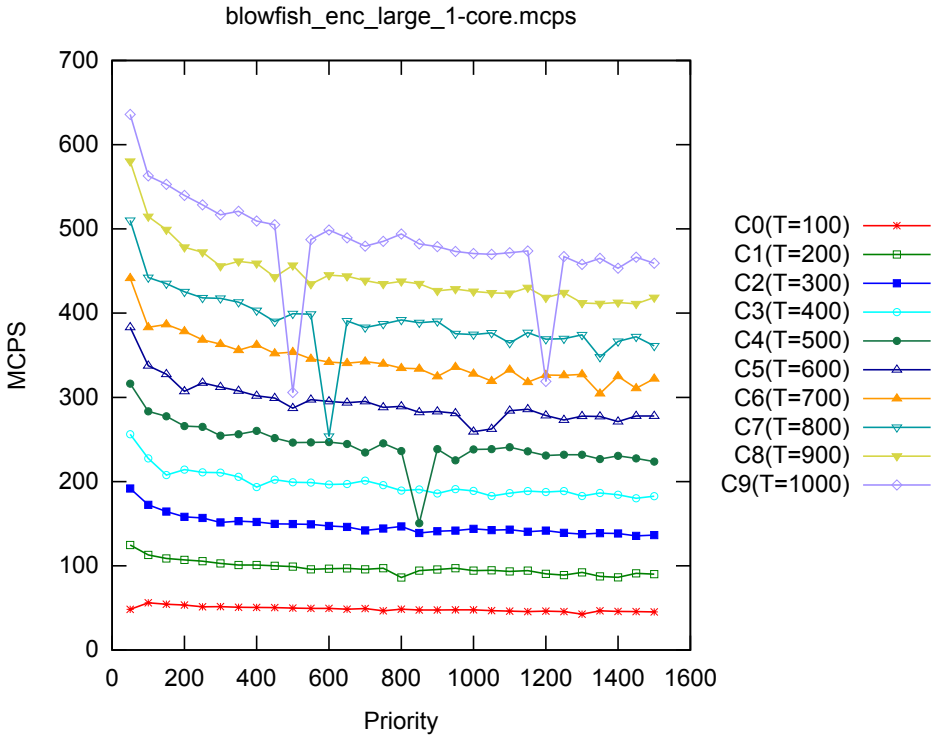


Figura 215: Desempenho em MCPS de Codificador Blowfish (Grande)

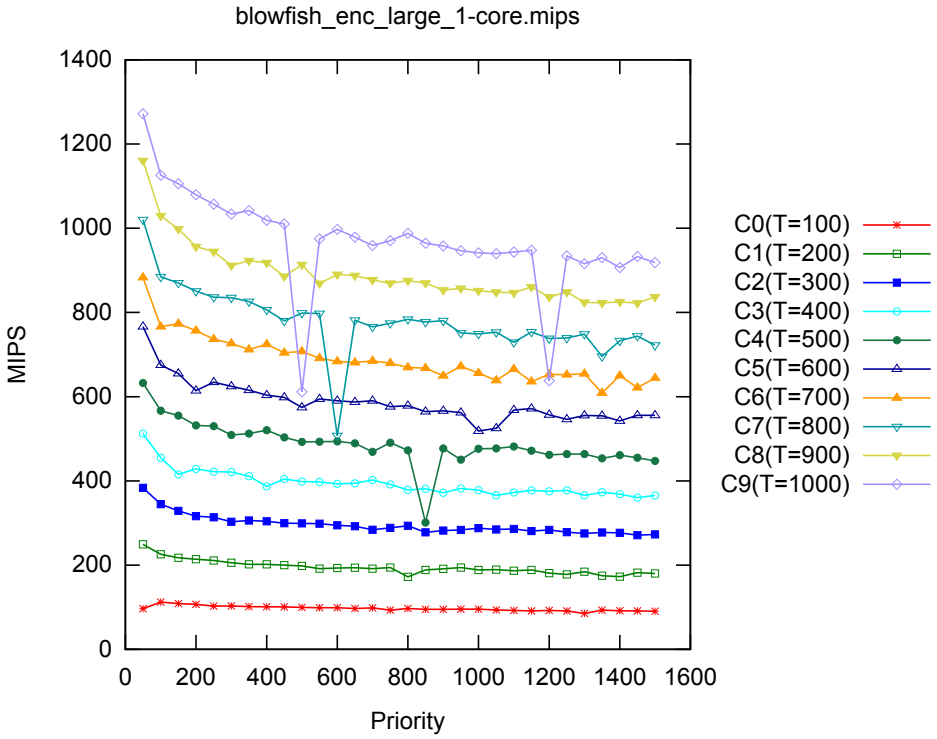


Figura 216: Desempenho em MIPS de Codificador Blowfish (Grande)

com erro relativo de 0,50% e desvio padrão de 3,230447265e+9 picosegundos (±2,13%). Com desempenho de 547,47 MCPS e 1.095 MIPS, o modelo proposto aumentou a velocidade simulação em cerca de 456 e 1.740 vezes, respecti-

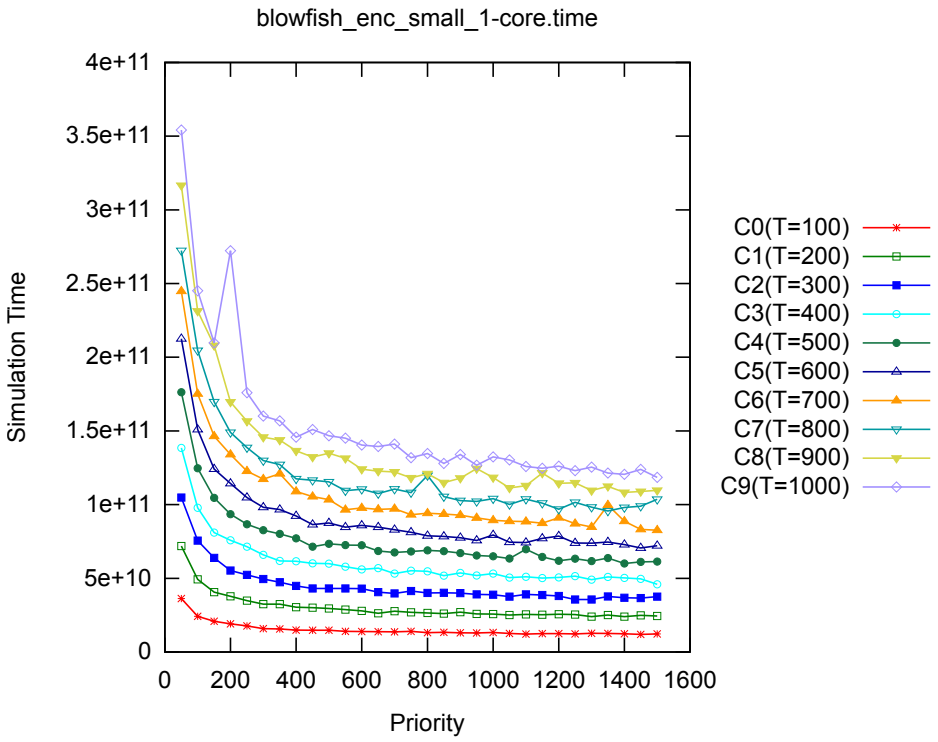


Figura 217: Tempo simulado de Codificador Blowfish (Pequeno)

vamente, quando comparado ao ISS.

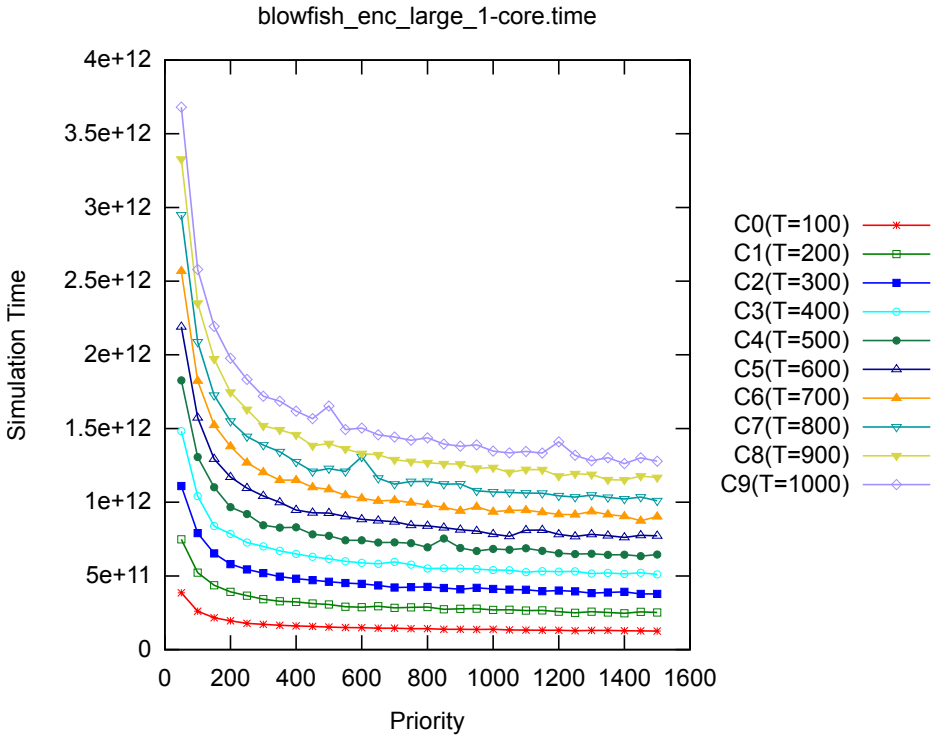


Figura 218: Tempo simulado de Codificador Blowfish (Grande)

Modificando a entrada para o tamanho grande, o algoritmo de codificação mantém o comportamento teórico e possui uma leve redução de erro

nas estimativas. Apesar dos desníveis entre as curvas, as variações detectadas são bem mais suaves do que as vistas anteriormente e vão permitir estimativas mais precisas. Na simulação com plataforma baseada em ISS foi obtido um tempo simulado de 1,578298786e+12 picosegundos e desempenho de 1,23 MCPS e 0,64 MIPS. Calibrando os parâmetros de ajuste e de prioridade do modelo proposto para os valores de 1.257 e 1.033, respectivamente, foi atingido um tempo simulado de 1,577258230114e+12 picosegundos em 9 simulações, com erro relativo ao ISS de 0,06% e desvio padrão de 9,946049275e+9 picosegundos (±0,63%). Foi obtido um desempenho de 554,43 MCPS e 1.109 MIPS e com estes resultados houve um aumento de desempenho de 451 e 1.722 vezes, respectivamente, quando comparado ao ISS.

### B.5.3 Rijndael Decoder

Originalmente chamado de Rijndael (87), este algoritmo de decodificação é mais conhecido como Advanced Encryption Standard (AES), sendo o padrão do governo norte americano para proteção de dados eletrônicos. Por ser um padrão e de grande relevância, o pacote de segurança do MiBench inclui o Rijndael em seu portfólio de aplicações, desenvolvendo entradas de tamanho pequeno e grande para realização das execuções.

Definindo uma parametrização de ajuste com 10 curvas (C0 até C9), variando entre 100 e 1.000, com intervalos de tamanho 100, e priorização com valores de 50 até 1.500, com passos de tamanho 50, são feitas as simulações com entrada de tamanho pequeno. Na figura 219 o desempenho em MCPS pode ser observado, com um pico de desempenho de cerca de 600 MCPS e, na figura 220, o desempenho em MIPS tem um pico de aproximadamente 1.200 MIPS. Estes resultados implicam em um aumento de velocidade de simulação de cerca de 546 e 1.846, respectivamente, uma vez que o modelo baseado em ISS obteve 1,10 MCPS e 0,65 MIPS de desempenho.

Seguindo a mesma parametrização de ajuste e de prioridade já definidas, os gráficos das figuras 221 e 222 ilustram o desempenho da decodificação de Rijndael com entrada de tamanho grande, para métricas de MCPS e MIPS, respectivamente. São observados picos de desempenho de cerca de 600 MCPS e 1.200 MIPS que representam uma melhoria de 541 e 1.846 vezes com relação ao desempenho do ISS que foi de 1,11 MCPS e 0,65 MIPS.

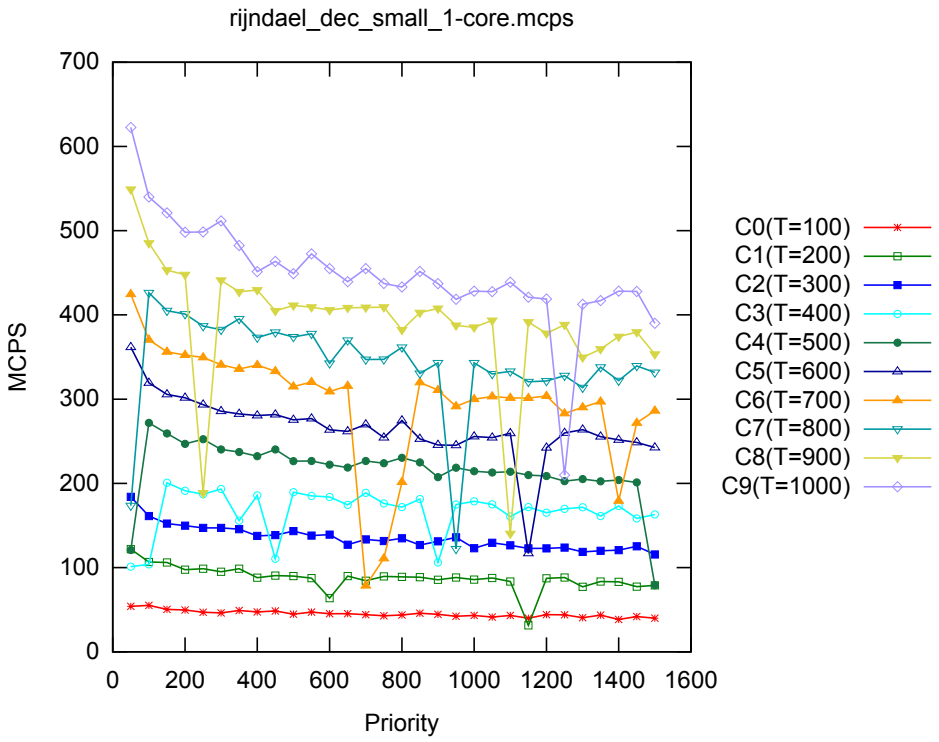


Figura 219: Desempenho em MCPS de Decodificador Rijndael (Pequeno)

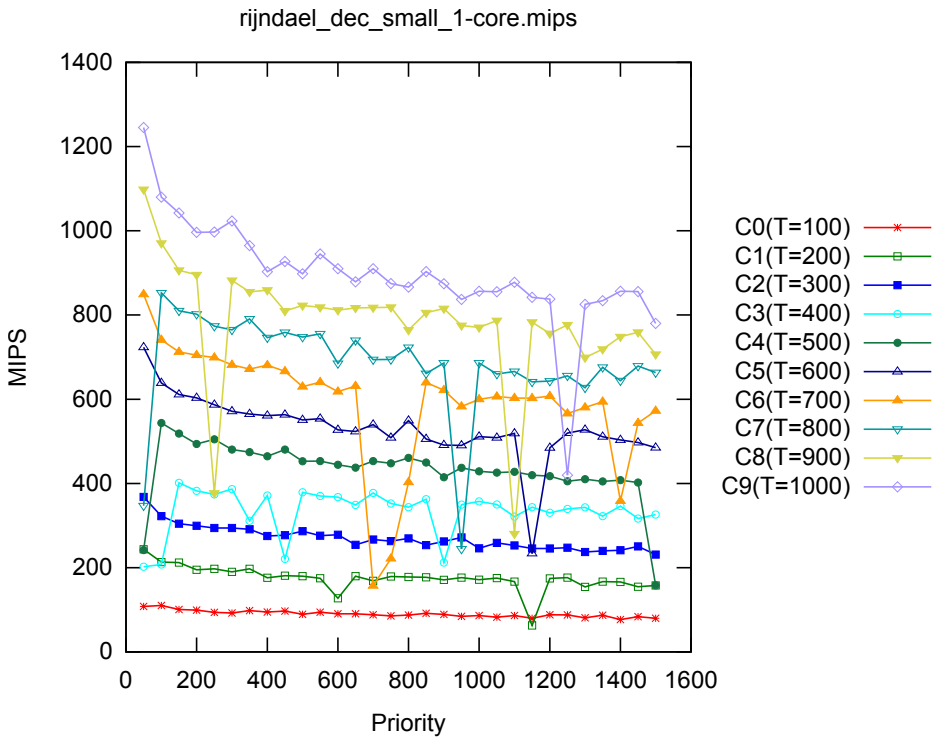


Figura 220: Desempenho em MIPS de Decodificador Rijndael (Pequeno)

Analisando as estimativas de tempo simulado para entrada de tamanho pequeno, os valores estimados podem ser visualizados na figura 223, seguindo a mesma parametrização já definida anteriormente. Em uma plataforma ba-

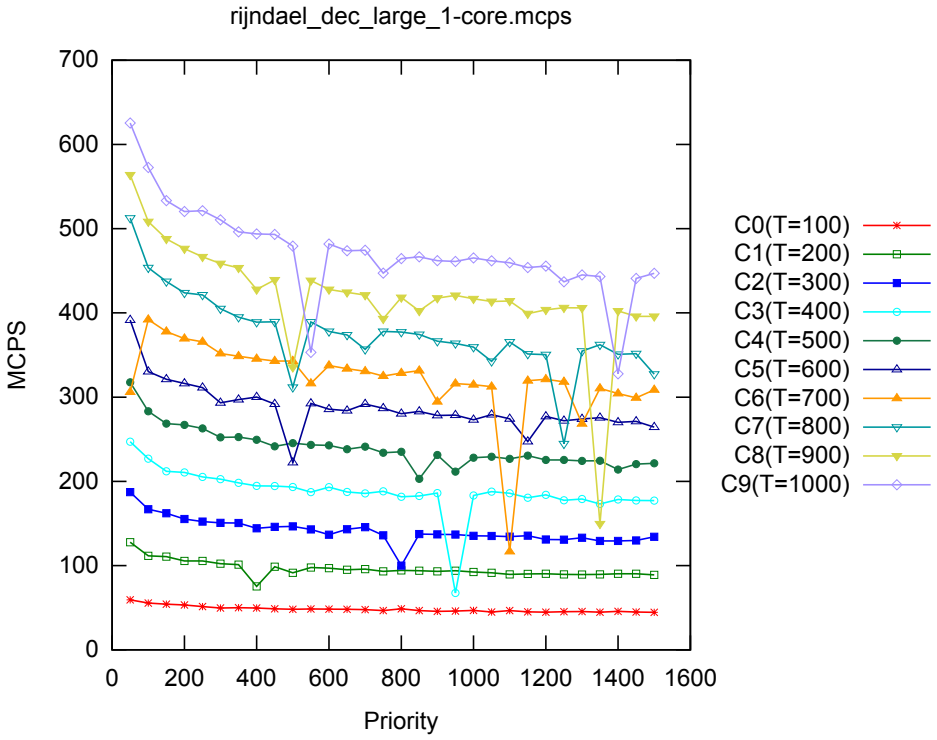


Figura 221: Desempenho em MCPS de Decodificador Rijndael (Grande)

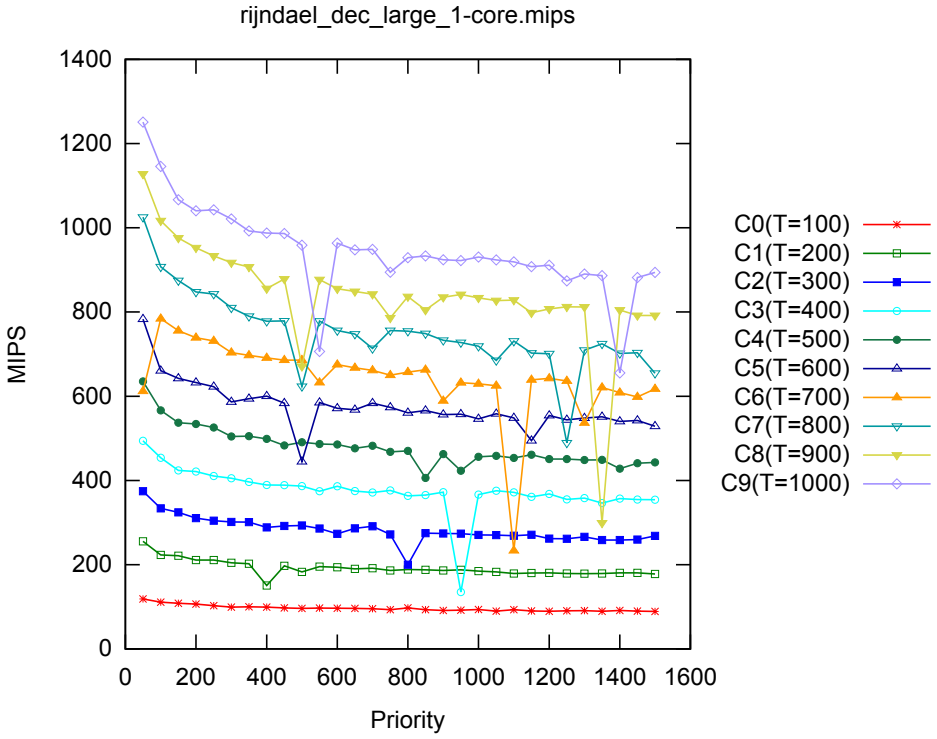


Figura 222: Desempenho em MIPS de Decodificador Rijndael (Grande)

seada em ISS, foi obtido um tempo simulado de 9,8901061e+10 picosegundos e um desempenho de simulação de 1,10 MCPS e 0,65 MIPS. Para que um comportamento o mais próximo possível seja observado no modelo proposto são

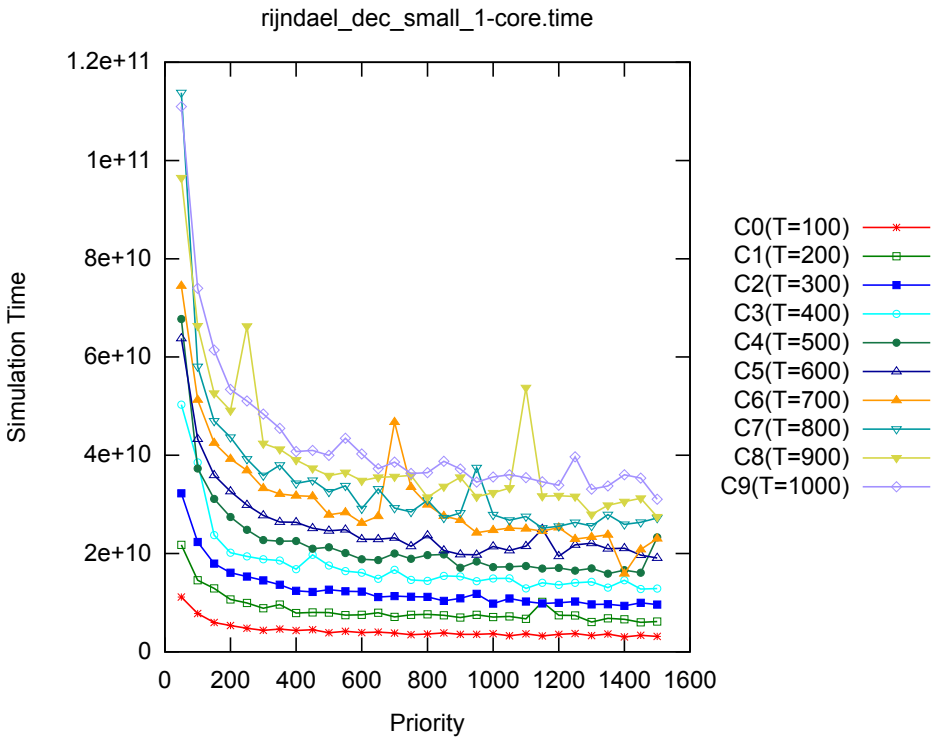


Figura 223: Tempo simulado de Decodificador Rijndael (Pequeno)

calibrados parâmetros de ajuste e de prioridade com valores de 3.021 e 1.031 que geram uma estimativa de tempo simulado de 9,9578160453e+10 picosegundos em 68 simulações, com erro relativo ao ISS de 0,68% e desvio padrão de 4,775586657e+9 picosegundos (±4,79%). O desempenho de simulação obtido foi de 1.210 MCPS e 2.420 MIPS que melhoram a velocidade de simulação da plataforma em cerca de 1.097 e 3.745 vezes, respectivamente, considerando o desempenho do ISS como referência.

Realizando simulações com entrada de tamanho grande e a mesma configuração de parâmetros são geradas as curvas vistas na figura 224. Com o aumento do tamanho da entrada foi melhorada a precisão das estimativas e isto se deve ao modelo teórico funcionar com mais precisão quando mais amostras são fornecidas. Na simulação utilizando o modelo baseado em ISS foi atingido um tempo simulado de 1,029889904e+12 picosegundos e desempenho de simulação de 1,11 MCPS e 0,65 MIPS. Para o modelo proposto se comportar de forma mais próxima possível são calibrados parâmetros de ajuste e de prioridade com valores de 2.989 e 1.114 que geram uma estimativa de tempo simulado de 1,036117754385e+12 picosegundos em 8 simulações. Os resultados mostram um desempenho de 1.273 MCPS e 2.546 MIPS que aumentam a velocidade da simulação em cerca de 1.143 e 3.903 vezes, respectivamente,

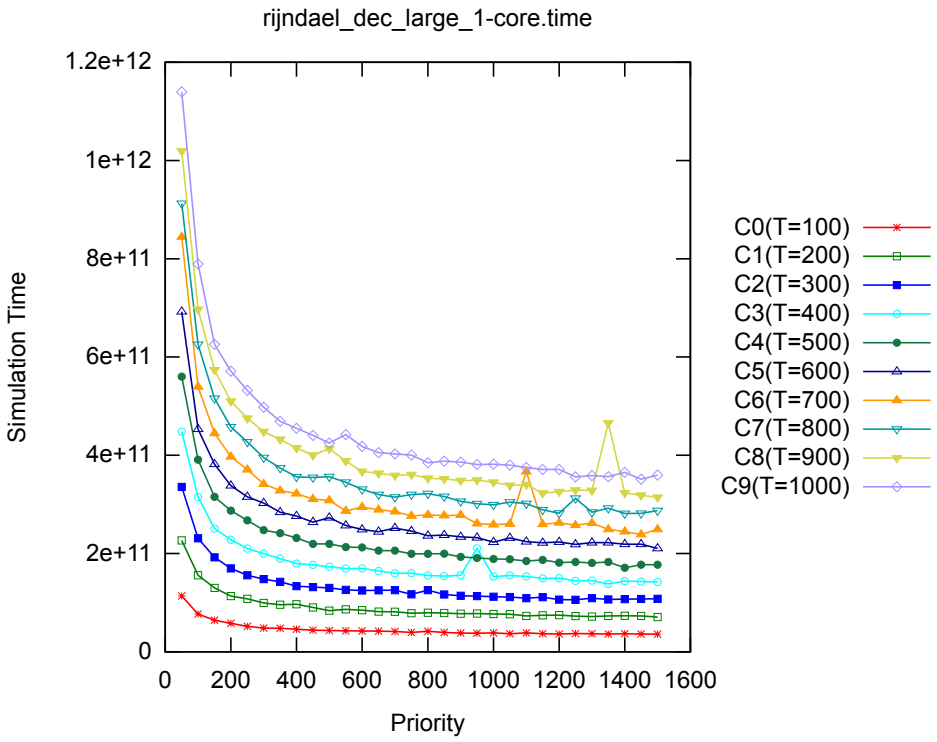


Figura 224: Tempo simulado de Decodificador Rijndael (Grande)

com um erro relativo 0,60% e desvio padrão de 1,0411517954e+10 picosegundos (±1,00%), com relação ao ISS.

### B.5.4 Rijndael Encoder

Mais conhecido pelo nome de Advanced Encryption System (AES), o algoritmo de codificação de Rijndael é atualmente o padrão do governo norte americano para proteção de informações digitais em seus sistemas. Este aplicativo está incluído no pacote de segurança do MiBench, como mais uma aplicação de grande representatividade na avaliação de desempenho de sistemas. Foram desenvolvidas duas entradas em arquivo de tamanho pequeno e grande que são carregados pela aplicação, sendo codificadas pelo algoritmo e tem as informações codificadas escritas em arquivos de saída.

Definindo uma parametrização de ajuste com 10 curvas (C0 até C9), com valores de 100 até 1.000 e intervalos de tamanho 100, e de prioridade com valores entre 50 e 1.500, com passos de tamanho 50, é feita a geração dos gráficos de desempenho em MCPS e MIPS, como pode ser visto nas figuras 225 e 226. Foi utilizada a entrada de tamanho pequeno e picos de desempenho de cerca de 550 MCPS e 1.100 MIPS foram observados, representando

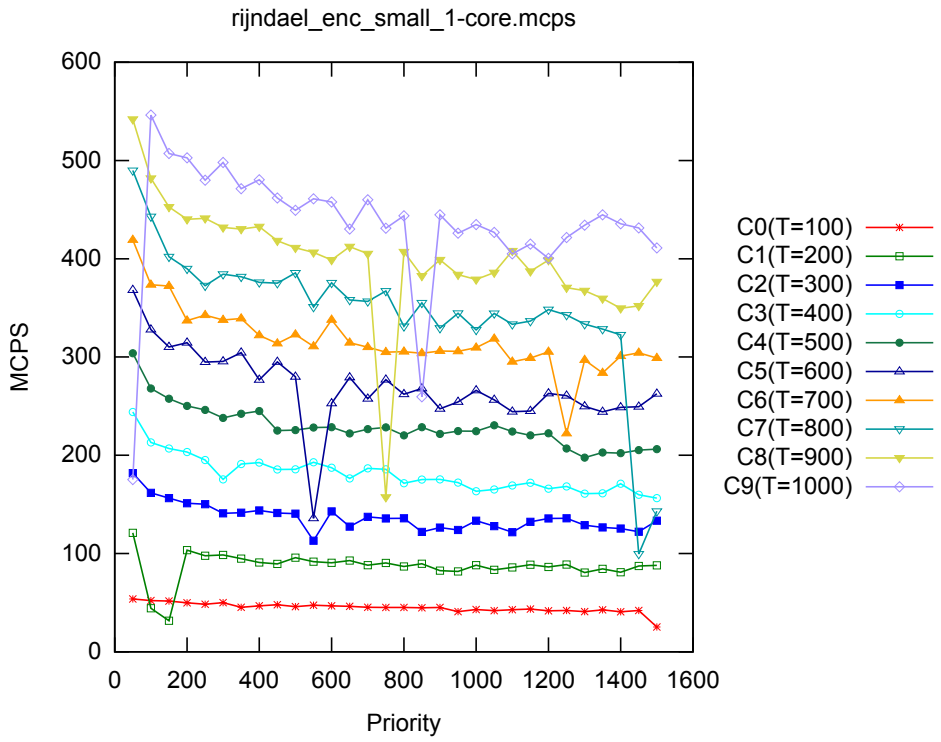


Figura 225: Desempenho em MCPS de Codificador Rijndael (Pequeno)

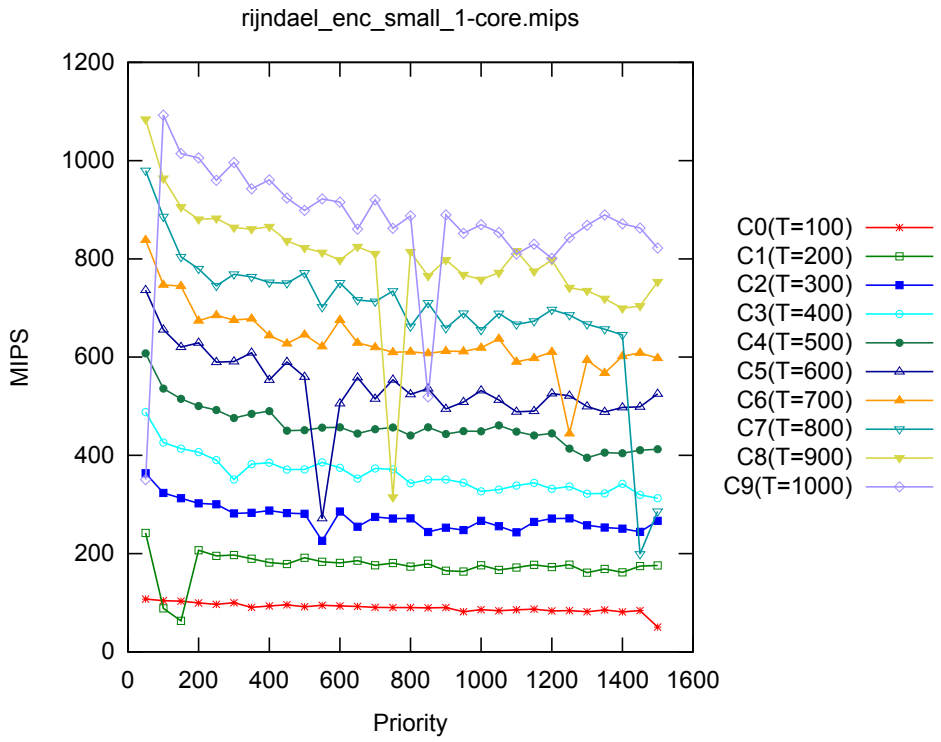


Figura 226: Desempenho em MIPS de Codificador Rijndael (Pequeno)

um aumento de desempenho de 496 e 1.692 vezes, respectivamente, quando comparada a execução no modelo ISS que obteve 1,11 MCPS e 0,65 MIPS.

Adotando a mesma parametrização de ajuste e de priorização descrita

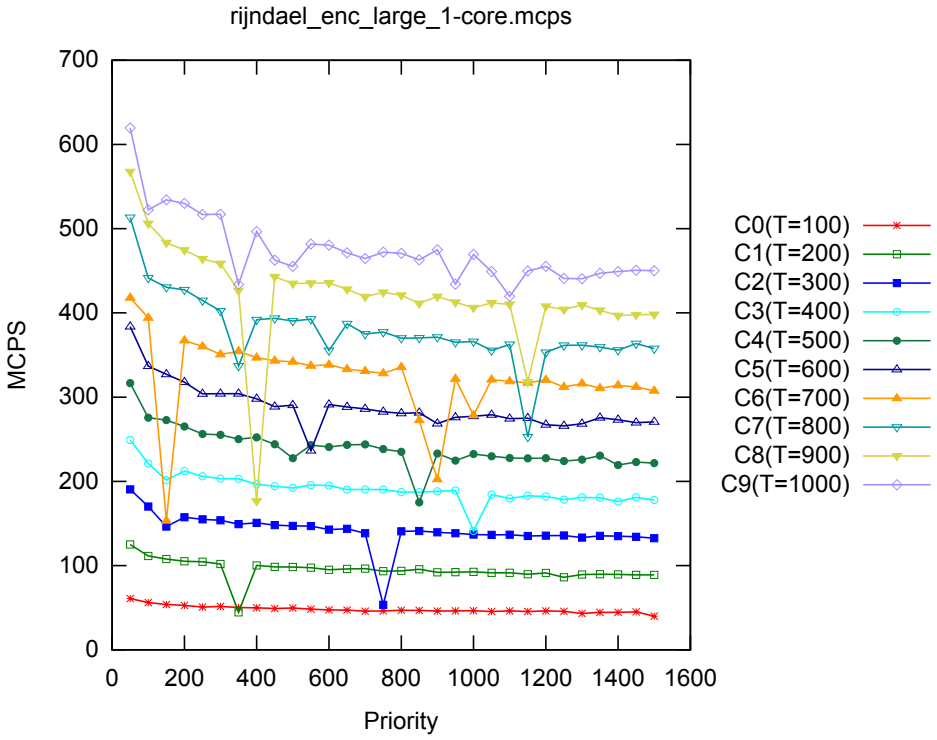


Figura 227: Desempenho em MCPS de Codificador Rijndael (Grande)

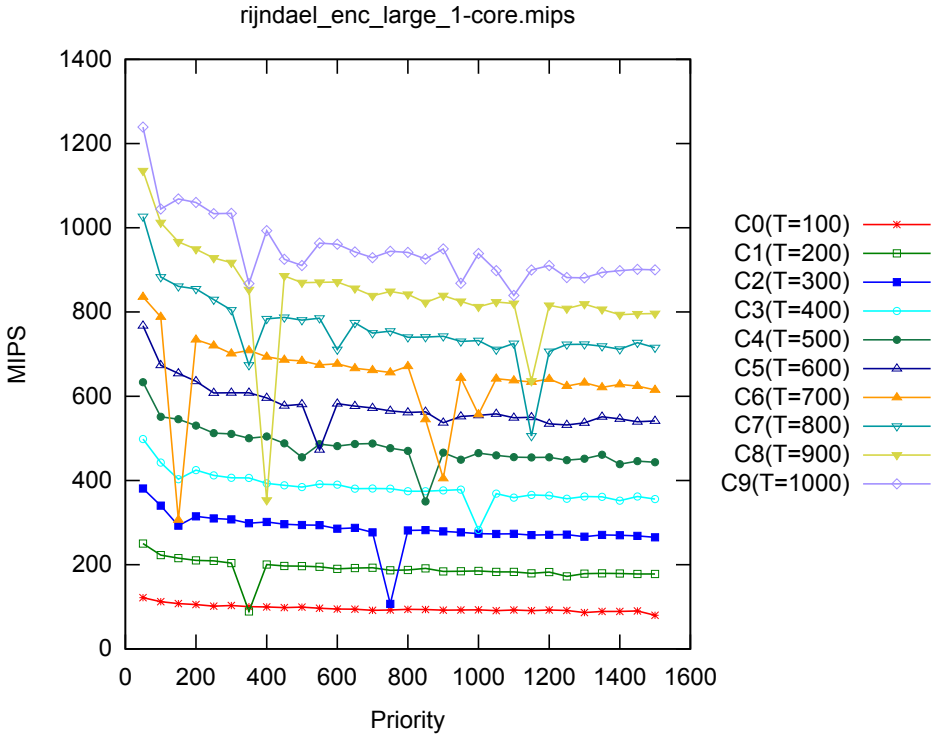


Figura 228: Desempenho em MIPS de Codificador Rijndael (Grande)

anteriormente, mas com uma entrada de tamanho grande, são gerados novos gráficos de desempenho que podem ser visualizados nas figuras 227 e 228. Podem ser observados picos de desempenho de cerca de 600 MCPS e 1.200

MIPS no modelo proposto que executa aproximadamente 541 e 1.846 vezes mais rápido que o modelo ISS com 1,11 MCPS e 0,65 MIPS de desempenho utilizado como referência.

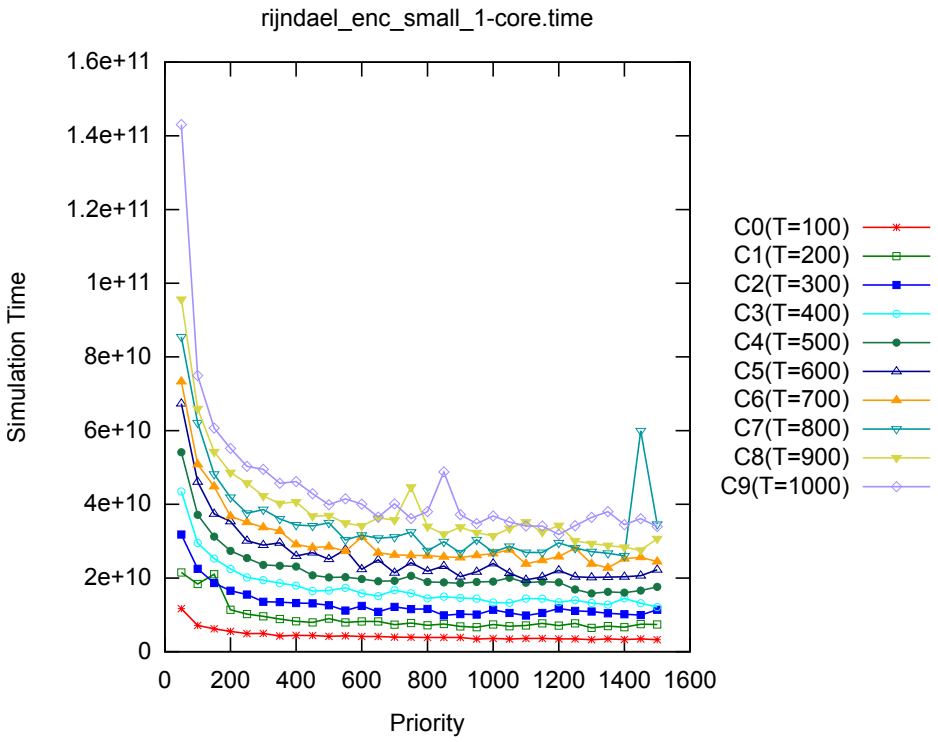


Figura 229: Tempo simulado de Codificador Rijndael (Pequeno)

Na análise das estimativas de tempo simulado geradas para entrada de tamanho pequeno, as curvas vistas na figura 229 ilustram o comportamento obtido, mais uma vez usando a parametrização já definida anteriormente. Simulando esta plataforma utilizando um modelo baseado em ISS foi obtido um tempo simulado de 9,8026836e+10 picosegundos e desempenho de simulação de 1,11 MCPS e 0,65 MIPS. Substituindo o modelo ISS pelo modelo proposto, procura-se obter um comportamento mais próximo possível do observado no ISS, e para tanto são calibrados parâmetros de ajuste e de priorização com valores de 2.886 e 921, respectivamente. Com esta configuração foi obtido um tempo simulado estimado de 9,9036306073e+10 picosegundos com 61 simulações e desempenho de 1.173 MCPS e 2.346 MIPS que melhoram o desempenho de execução em 1.055 e 3.587 vezes, com erro relativo ao ISS de 1,02% e desvio padrão de 5,156280674e+9 picosegundos (±5,20%).

Aplicando o conjunto de entrada de tamanho grande, como visto na figura 230, é possível notar os efeitos dos ruídos são atenuados e o comportamento teórico previsto se apresenta com maior qualidade em suas estimativas.

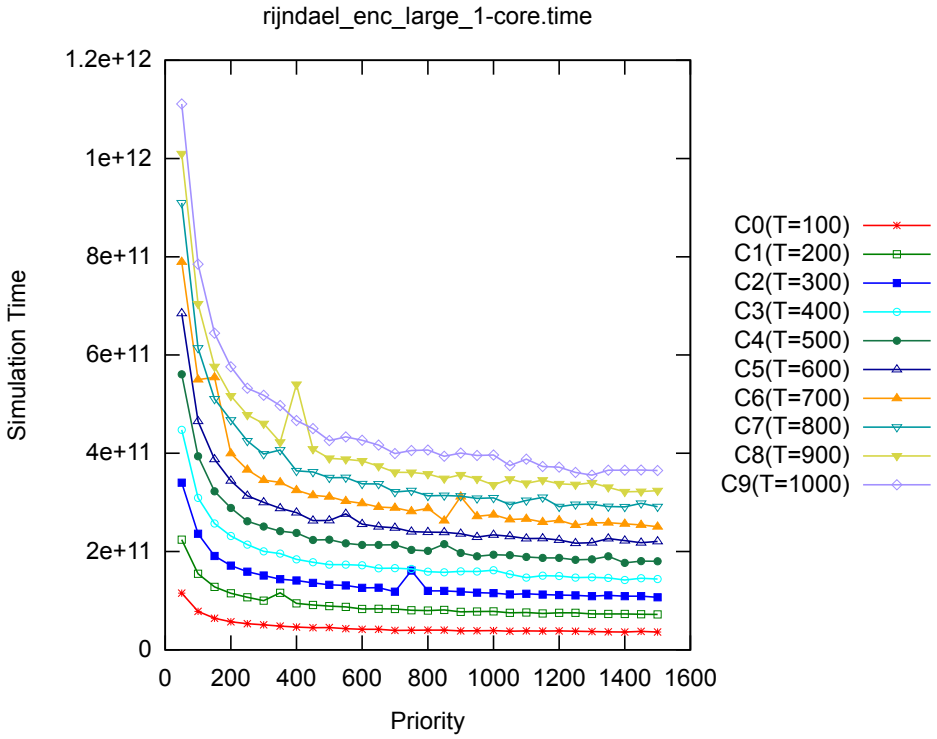


Figura 230: Tempo simulado de Codificador Rijndael (Grande)

Com o modelo de ISS e este tamanho de entrada foi obtido um tempo simulado de 1,020808558e+12 picosegundos e desempenho de execução da simulação de 1,11 MCPS e 0,65 MIPS. Para obter um comportamento equivalente são calibrados parâmetros de ajuste e de prioridade com valores de 2.939 e 1.153 que geram uma estimativa de tempo simulado de 1,024353411192e+12 picosegundos em 23 simulações, com erro relativo ao ISS de 0,34% e desvio padrão de 1,4318205005e+10 picosegundos (±1,39%). O desempenho obtido pelo modelo proposto foi de 1.252 MCPS e 2.503 MIPS que representam um aumento de 1.129 e 3.841 vezes da velocidade de simulação obtida utilizando o modelo ISS.

### B.5.5 SHA

A sigla SHA vem do inglês Secure Hash Algorithm (88) e este algoritmo em particular é um padrão do governo federal norte americano para geração de chaves hash. Uma função hash consiste em transformar uma entrada de tamanho arbitrário em uma chave de tamanho fixo, de tal forma que seja impossível obter a entrada original utilizada. O aplicativo SHA está incluso no pacote de segurança do MiBench, com duas entradas de tamanho pequeno

e grande foram especialmente desenvolvidas para que possa ser feita uma análise da execução do algoritmo sob condições de diferentes tamanhos de entrada e fornecer informações mais detalhadas do comportamento.

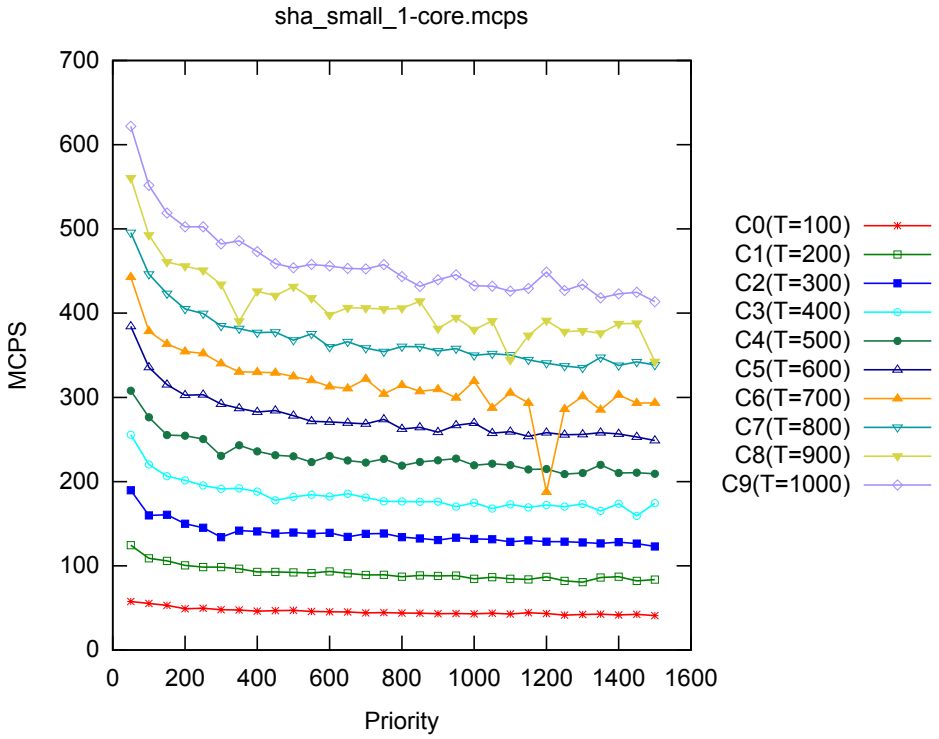


Figura 231: Desempenho em MCPS de SHA (Pequeno)

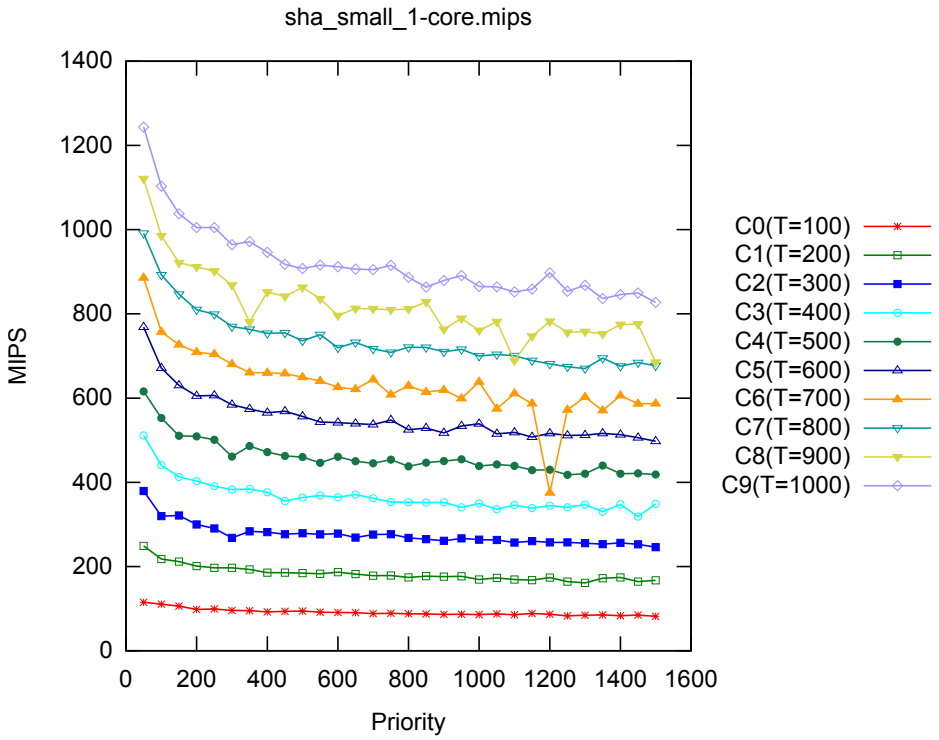


Figura 232: Desempenho em MIPS de SHA (Pequeno)

Definindo uma parametrização de 10 curvas de ajuste (C0 até C9), com valores entre 100 e 1.000, com intervalos de tamanho 100, e valores de prioridade de 50 até 1.500, com passos de tamanho 50, são gerados os gráficos de desempenho para entrada de tamanho pequeno. Os valores são exibidos nas métricas de MCPS e MIPS e podem ser visualizados nas figuras 231 e 232, respectivamente. Podem ser vistos picos de desempenho de cerca de 600 MCPS e 1.200 MIPS que representam uma melhoria de aproximadamente 488 e 1.846 vezes, comparado ao desempenho do ISS que foi de 1,23 MCPS e 0,65 MIPS.

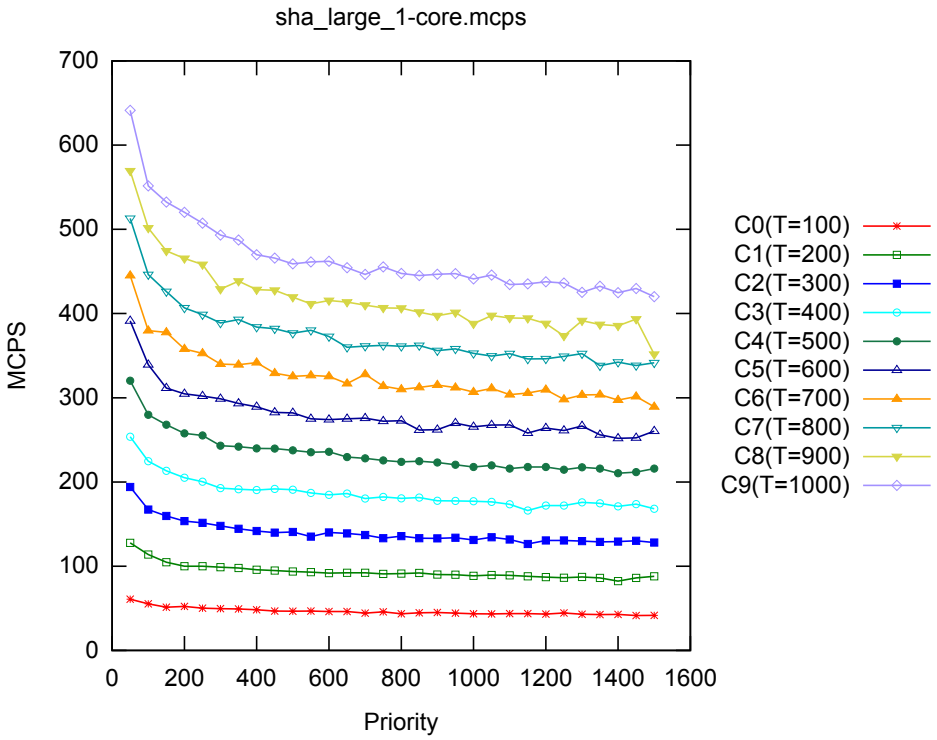


Figura 233: Desempenho em MCPS de SHA (Grande)

Utilizando a mesma parametrização descrita anteriormente, são feitas diversas simulações com o modelo proposto com entrada de tamanho grande. Os resultados de desempenho gerados podem ser visualizados nas figuras 233 e 234, considerando métricas em MCPS e MIPS, respectivamente. Pode ser visto picos de desempenho de aproximadamente 650 MCPS e 1.300 MIPS, implicando em aumento de desempenho de cerca de 516 e 1.970 vezes, quando comparado ao ISS que obteve 1,26 MCPS e 0,66 MIPS.

Na figura 235, são exibidas as curvas de estimativas de tempo simulado geradas utilizando a parametrização definida. O modelo proposto se comporta conforme o previsto e apesar de pequenos desvios da função teórica original, uma vez calculadas as constantes do modelo de tempo, o projetista

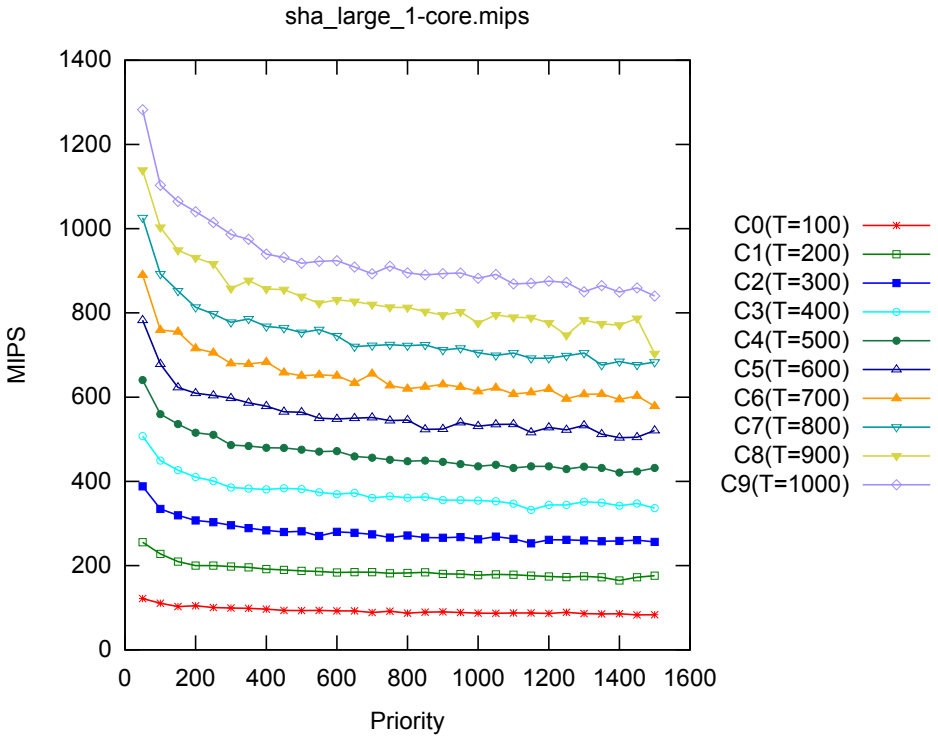


Figura 234: Desempenho em MIPS de SHA (Grande)

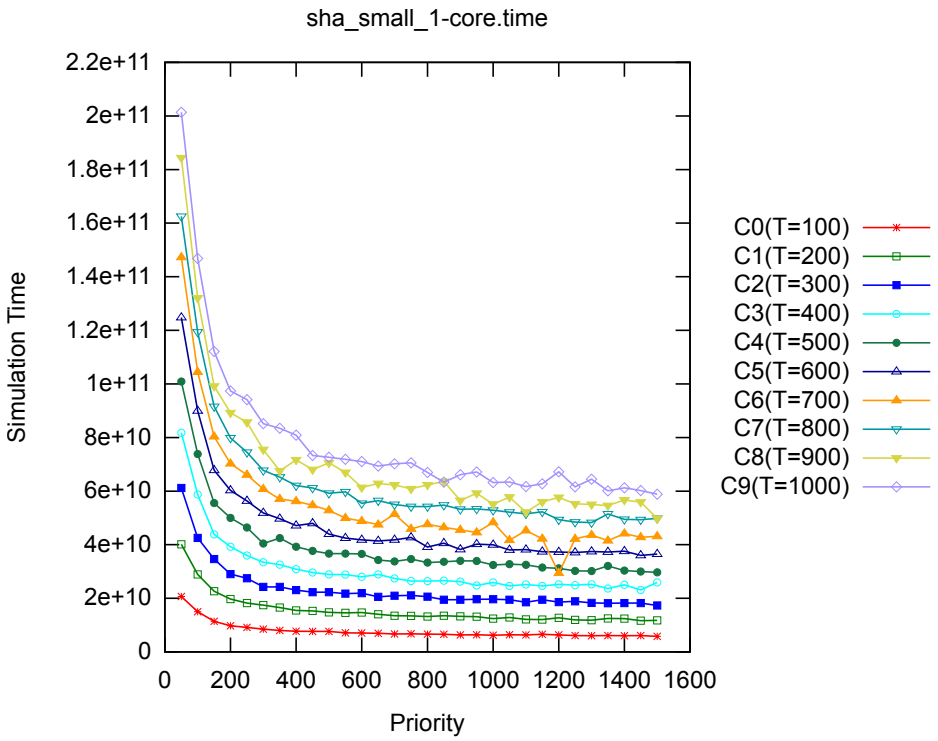


Figura 235: Tempo simulado de SHA (Pequeno)

é capaz de realizar previsões estáticas sobre o sistema. Em uma implementação da plataforma utilizando um modelo ISS foi obtido um tempo simulado de 6,0487355e+10 picosegundos, com 1,23 MCPS e 0,65 MIPS de desempenho

atingido. Como o objetivo é equiparar o comportamento obtido com ISS, medindo o ganho de desempenho e o erro associado, foram calibrados parâmetros de ajuste e de priorização com valores de 988 e 958, respectivamente, que geraram uma estimativa de tempo simulado de 5,9643313507e+10 picosegundos em 32 simulações. Foi obtido também um desempenho de 397,61 MCPS e 795,23 MIPS que representam um aumento de 323 e 1.232 vezes do desempenho do ISS, com erro relativo de 1,39% e desvio padrão de 1,595975908e+9 picosegundos (±2,67%).

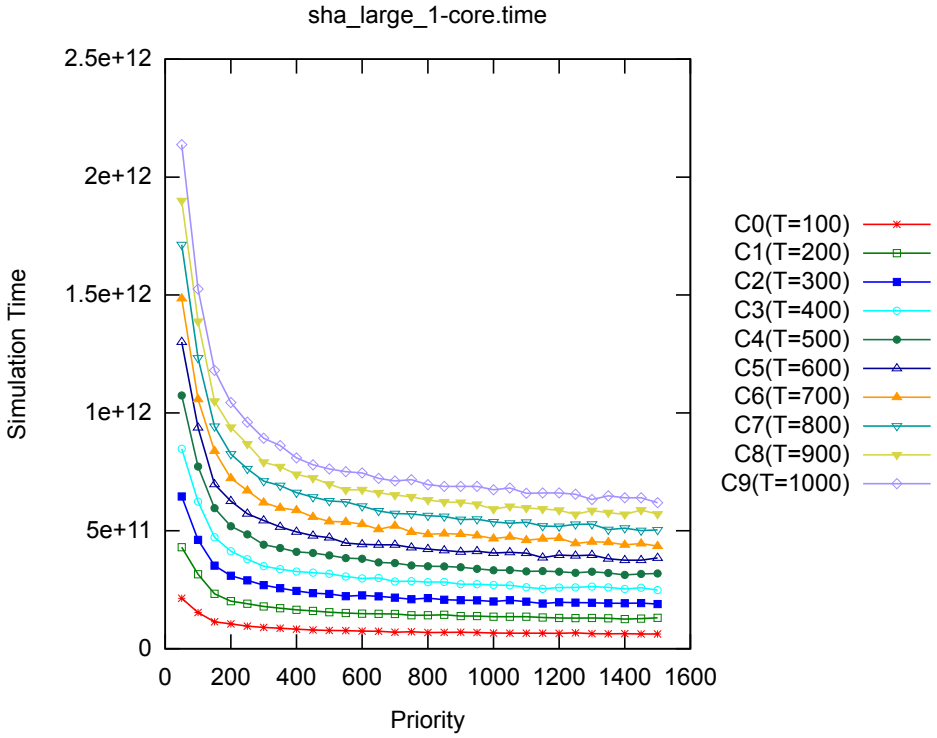


Figura 236: Tempo simulado de SHA (Grande)

Durante as análises de estimativas de tempo simulado para entrada de tamanho grande, que podem ser visualizadas na figura 236, foi observado mais uma vez o comportamento teórico previsto e uma melhoria significativa na qualidade das estimativas geradas. Na simulação utilizando ISS foi obtido um tempo simulado de 6,29603739e+11 picosegundos e desempenho de 1,26 MCPS e 0,66 MIPS. Substituindo o ISS pelo modelo proposto e com o objetivo de equiparar o comportamento observado, são calibrados parâmetros de ajuste e de prioridade de com valores de 1.033 e 1.134, respectivamente, que geram uma estimativa de tempo simulado de 6,33915175734e+11 picosegundos em 6 simulações. Foi obtido um desempenho de 418,51 MCPS e 837,02 MIPS que melhoram o desempenho com ISS em 333 e 1.269 vezes, com erro relativo de

0,68% e desvio padrão de 4,805952408e+9 picosegundos (±0,75%).

## B.6 MiBench Telecomm

Para representar as aplicações de telecomunicações, o pacote MiBench inclui uma série de aplicações de grande relevância para o funcionamento dos sistemas de comunicação, permitindo que o desempenho da plataforma em questão seja avaliado neste contexto de aplicação.

### B.6.1 ADPCM Decoder

Este algoritmo para decodificação é muito utilizado em sistemas de telefonia, permitindo a compressão do sinal através da modulação adaptativa diferencial do pulso de código ou Adaptative Differential Pulse-Code Modulation (ADPCM) (89). Sua implementação em software está incluída no pacote de telecomunicações do MiBench, com entradas para serem processadas na codificação ADPCM com tamanhos pequeno e grande, sendo fornecidas através de leitura de arquivos para sua decodificação.

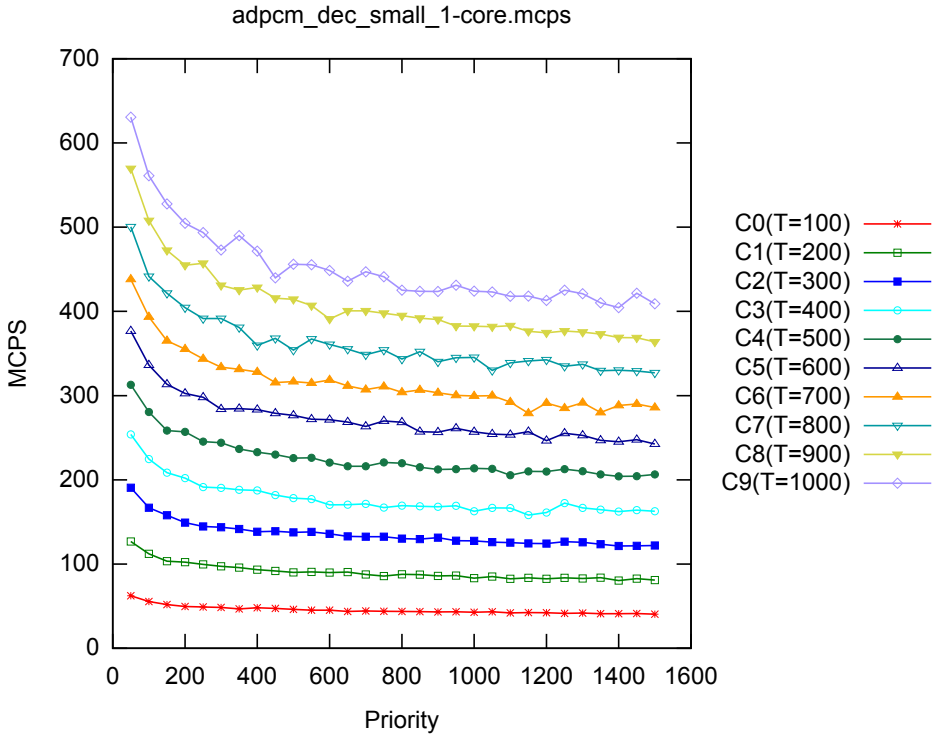


Figura 237: Desempenho em MCPS de Decodificador ADPCM (Pequeno)

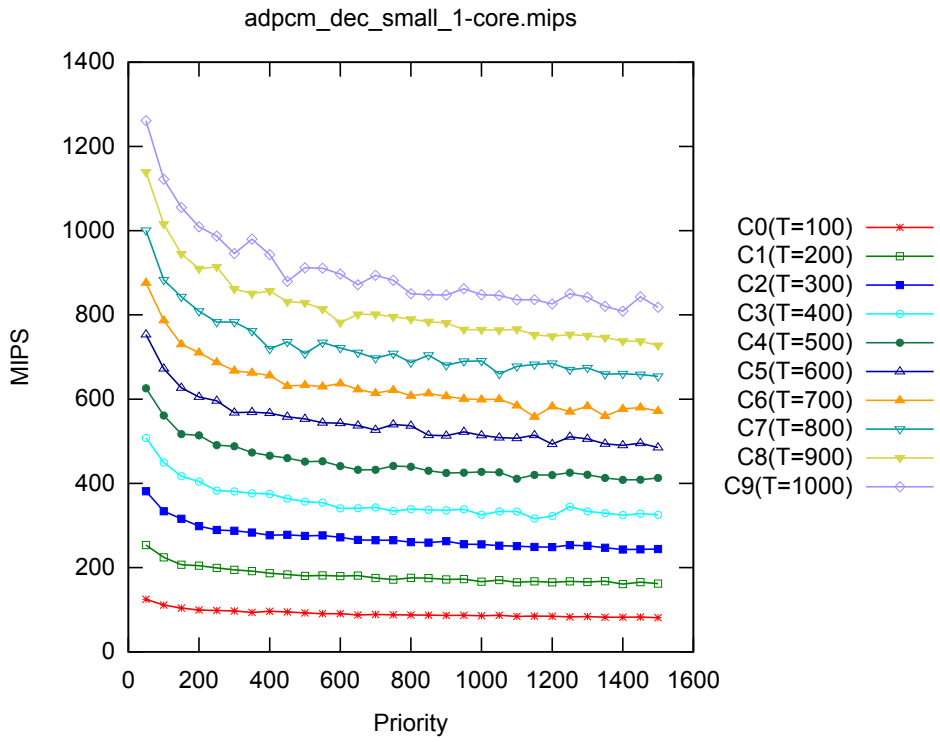


Figura 238: Desempenho em MIPS de Decodificador ADPCM (Pequeno)

Para realização dos experimentos foi definida uma parametrização de ajuste com 10 curvas (C0 até C9), com valores de 100 até 1.000 e intervalos de tamanho 100, e de prioridade com valores que vão de 50 até 1.500 e passos de tamanho 50. Utilizando a entrada de tamanho pequeno, foram gerados os gráficos de desempenho vistos nas figuras 237 e 238. Os picos de cerca de 600 MCPS e 1.200 MIPS representam uma melhoria de 480 e 1.818 vezes, respectivamente, do desempenho obtido no ISS que foi de 1,25 MCPS e 0,66 MIPS.

Adotando a mesma configuração de parâmetros definida anteriormente, é utilizada a entrada de tamanho grande e é feita a geração dos gráficos vistos nas figuras 239 e 240, com métricas em MCPS e MIPS, respectivamente. Foram observados picos de desempenho de aproximadamente 600 MCPS e 1.200 MIPS que aumentam a velocidade de simulação em 472 e 1.791 vezes, respectivamente, uma vez que o ISS obteve 1,27 MCPS e 0,67 MIPS de desempenho.

Para analisar o comportamento e qualidade das estimativas de tempo simulado realizadas, considerando entrada de tamanho pequeno, é feita a geração das curvas vistas na figura 241. É possível observar que o comportamento teórico previsto foi confirmado e que as estimativas apresentam baixo

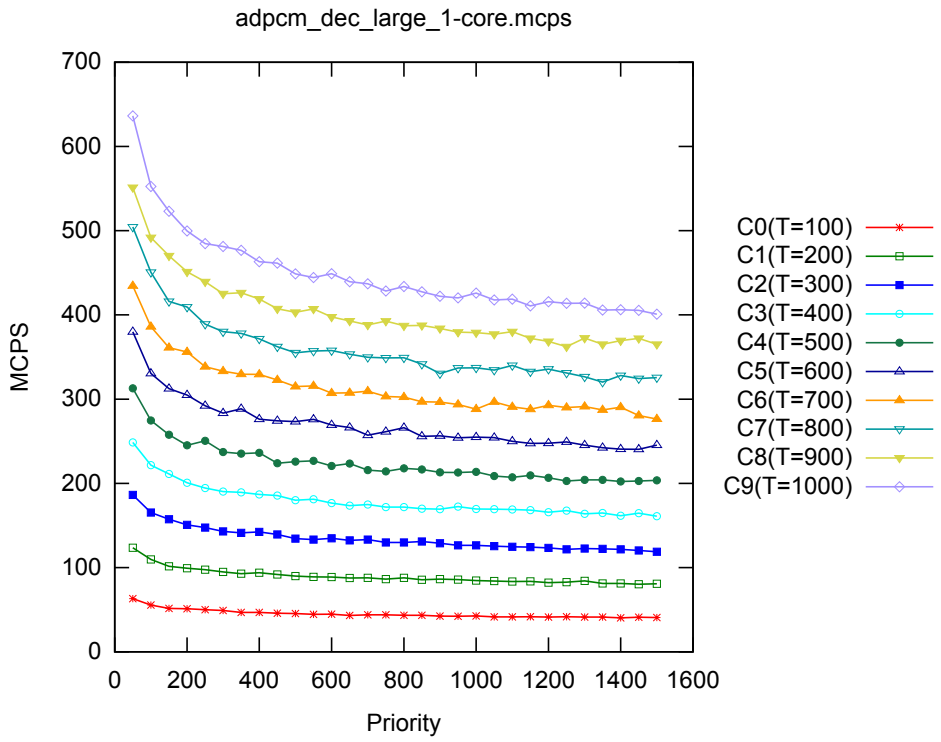


Figura 239: Desempenho em MCPS de Decodificador ADPCM (Grande)

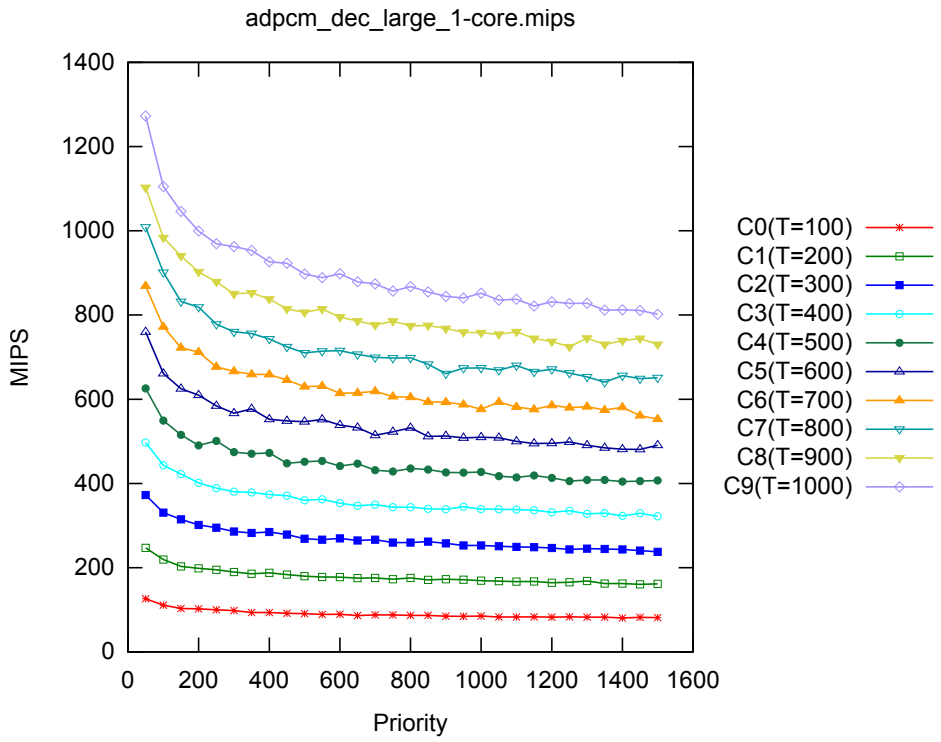


Figura 240: Desempenho em MIPS de Decodificador ADPCM (Grande)

índices de erro associados, mesmo com a entrada de tamanho reduzido. Executando o decodificador ADPCM em uma plataforma baseada em ISS foi atingido um tempo simulado de 1,34463075e+11 picosegundos e desempenho de

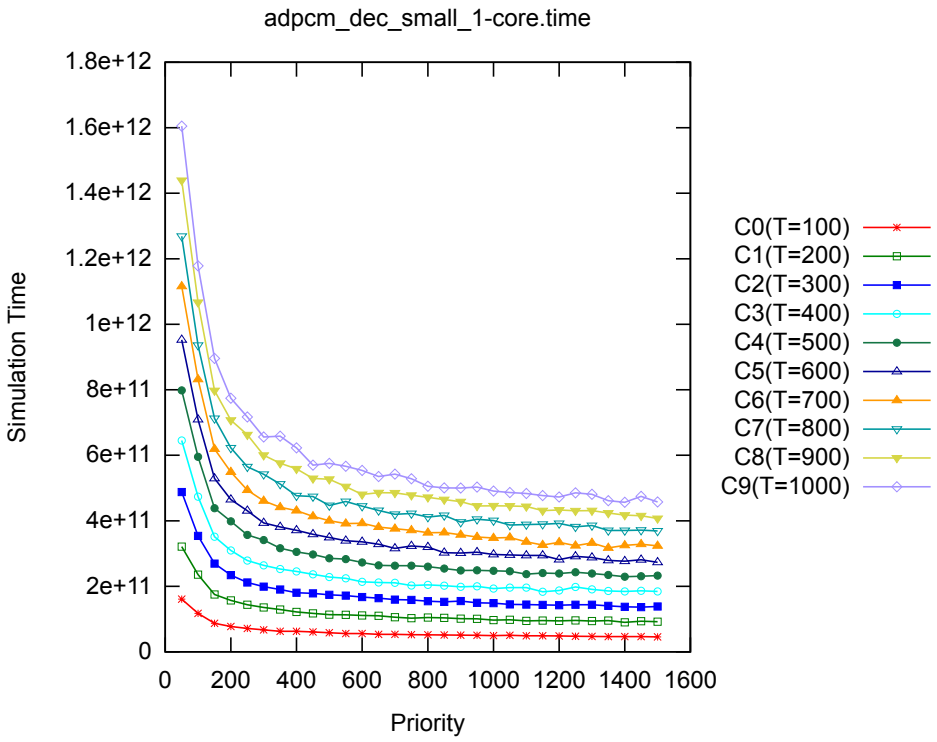


Figura 241: Tempo simulado de Decodificador ADPCM (Pequeno)

simulação de 1,25 MCPS e 0,66 MIPS. Para obter um comportamento equivalente no modelo proposto, foram calibrados parâmetros de ajuste e de prioridade com valores de 297 e 1.042 que geraram uma estimativa de tempo simulado de 1,3391264264e+11 picosegundos em 5 simulações, com erro relativo ao ISS de 0,40% e desvio padrão de 2,151452035e+9 picosegundos (±1,60%). O desempenho obtido foi de 116,67 MCPS e 233,34 MIPS que significam uma melhoria de 94 e 356 vezes, respectivamente, da velocidade de simulação, quando comparado ao modelo ISS.

Em simulação utilizando modelo ISS, com entrada de tamanho grande, foi obtido um tempo simulado de 2,658696919e+12 picosegundos e resultados de simulação com desempenho de 1,27 MCPS e 0,67 MIPS. Para equiparar o comportamento visto no ISS, o modelo proposto utiliza parâmetros de ajuste e de prioridade com valores de 303 e 1.098 para gerar uma estimativa de tempo simulado de 2,640710358332e+12 picosegundos em 5 simulações, com desempenho de 119,84 MCPS e 239,67 MIPS. Apresentando erro relativo ao ISS de 0,67% e desvio padrão de 9,338220443e+9 picosegundos (±0,35%), o desempenho obtido na simulação melhora em cerca de 95 e 360 vezes, respectivamente, quando comparado ao ISS em métricas MCPS e MIPS.

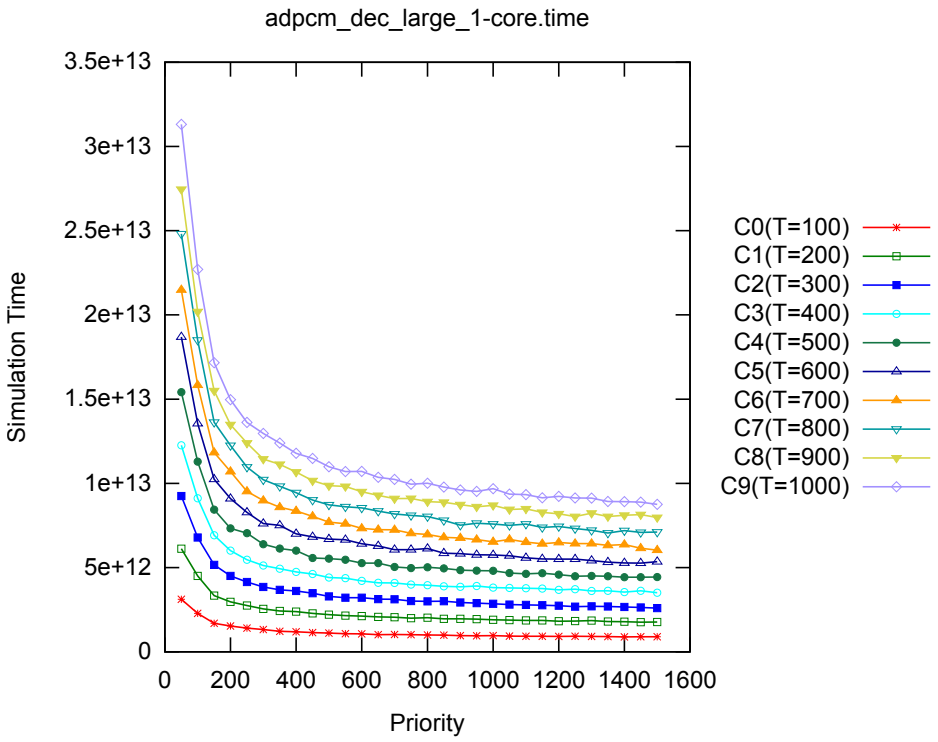


Figura 242: Tempo simulado de Decodificador ADPCM (Grande)

### B.6.2 ADPCM Encoder

O Adaptative Differential Pulse-Code Modulation ou ADPCM é uma técnica de compressão muito utilizada em sistemas de telefonia, possuindo variantes u-law, que é adotada nos EUA e no Japão, e A-law, que é utilizada na Europa e no resto do mundo. Uma implementação em software do codificador ADPCM está incluída no pacote de telecomunicações do MiBench, com duas entradas de tamanho pequeno e grande que são carregadas de arquivos durante a execução da aplicação, gerando o resultado codificado.

Configurando os parâmetros de ajuste para que sejam geradas 10 curvas (C0 até C9), com valores entre 100 e 1.000, e intervalos de tamanho 100, e de prioridade com valores entre 50 e 1.500, com intervalos de tamanho 50, são gerados gráficos de desempenho para entrada de tamanho pequeno, como pode ser visto nas figuras 243 e 244. Nas métricas de MCPS e MIPS são obtido picos de desempenho de cerca de 600 MCPS e 1.200 MIPS com ganho de 472 e 1.875 vezes, quando comparado ao ISS que obteve 1,27 MCPS e 0,64 MIPS.

Adotando a mesma parametrização descrita anteriormente, são feitas análises de desempenho para entrada de tamanho grande e são gerados os gráficos vistos nas figuras 245 e 246. No modelo ISS é atingido 1,35 MCPS e 0,68 MIPS

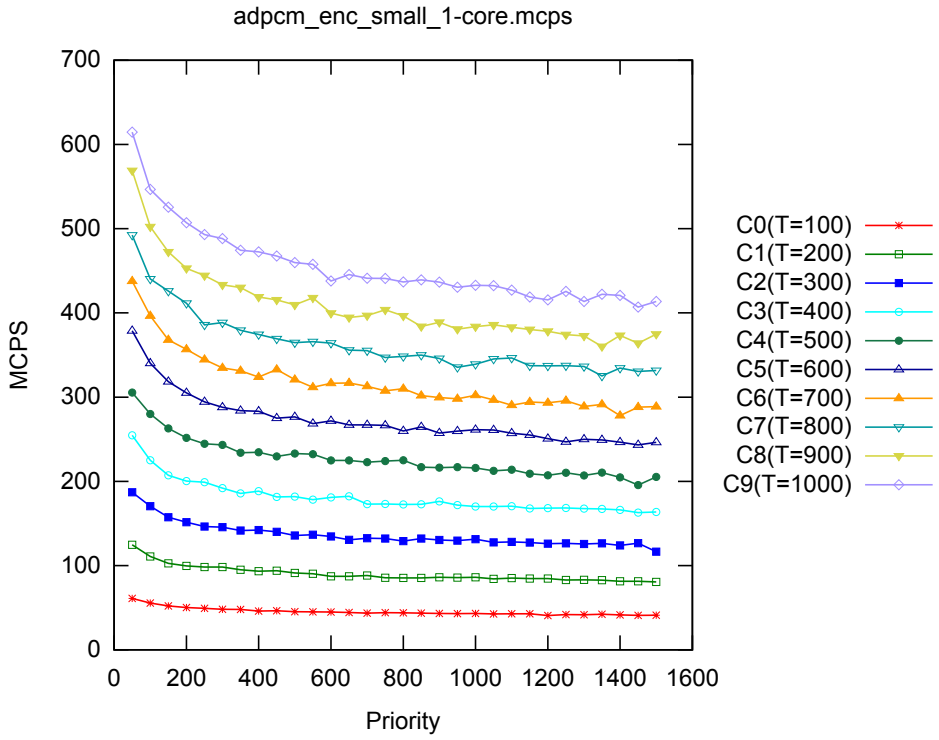


Figura 243: Desempenho em MCPS de Codificador ADPCM (Pequeno)

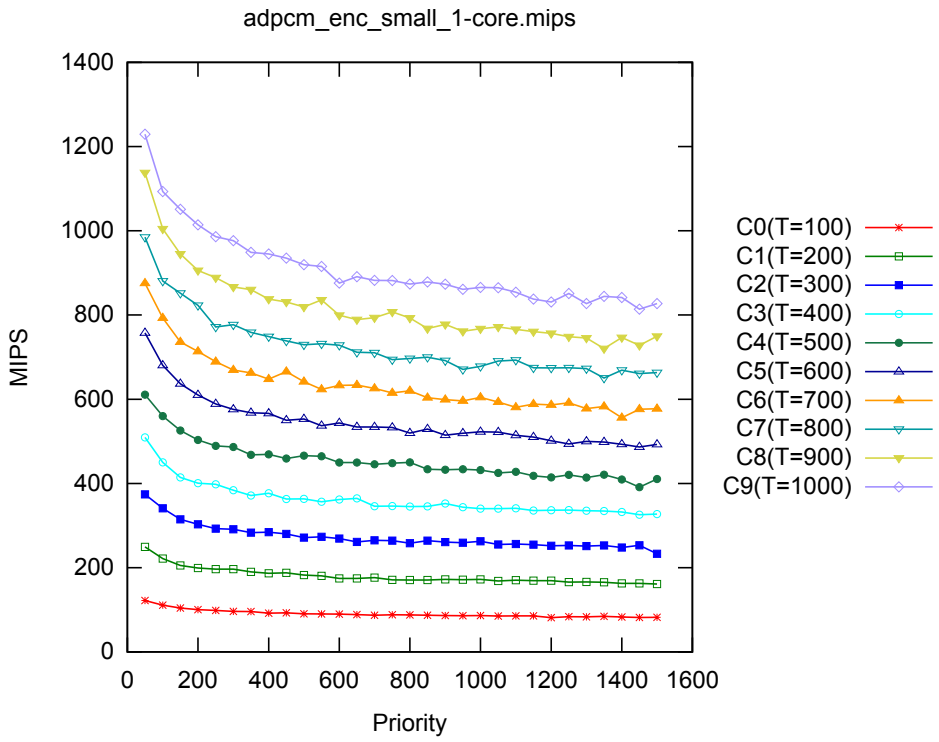


Figura 244: Desempenho em MIPS de Codificador ADPCM (Pequeno)

de desempenho, enquanto que o modelo proposto atinge picos de cerca de 600 MCPS e 1.200 MIPS que aumentam o desempenho relativo ao ISS em 444 e 1.765 vezes, respectivamente.

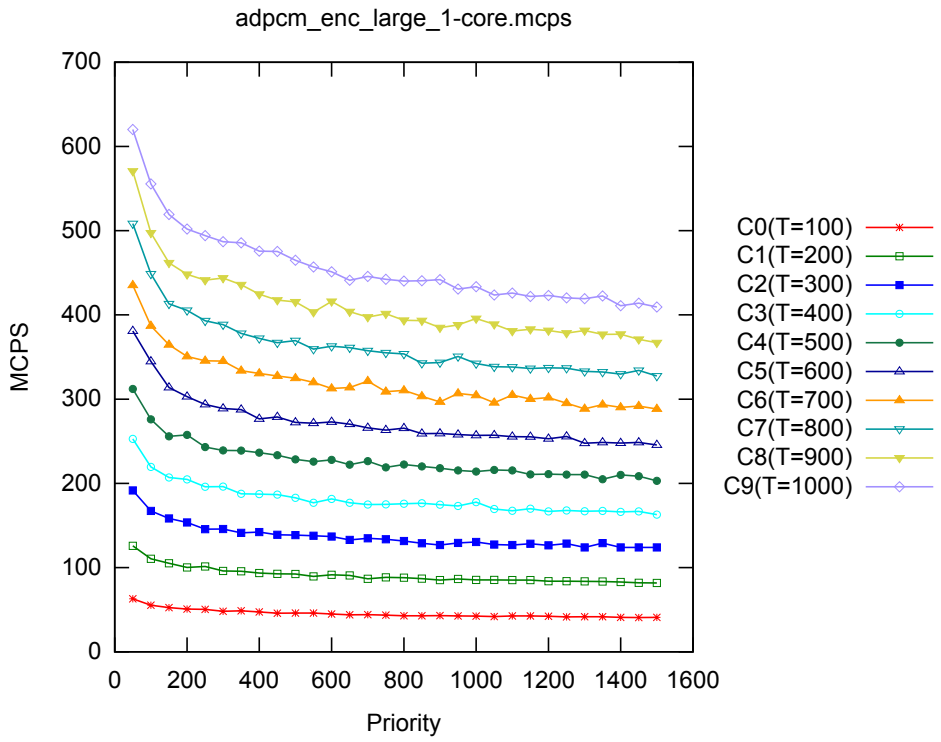


Figura 245: Desempenho em MCPS de Codificador ADPCM (Grande)

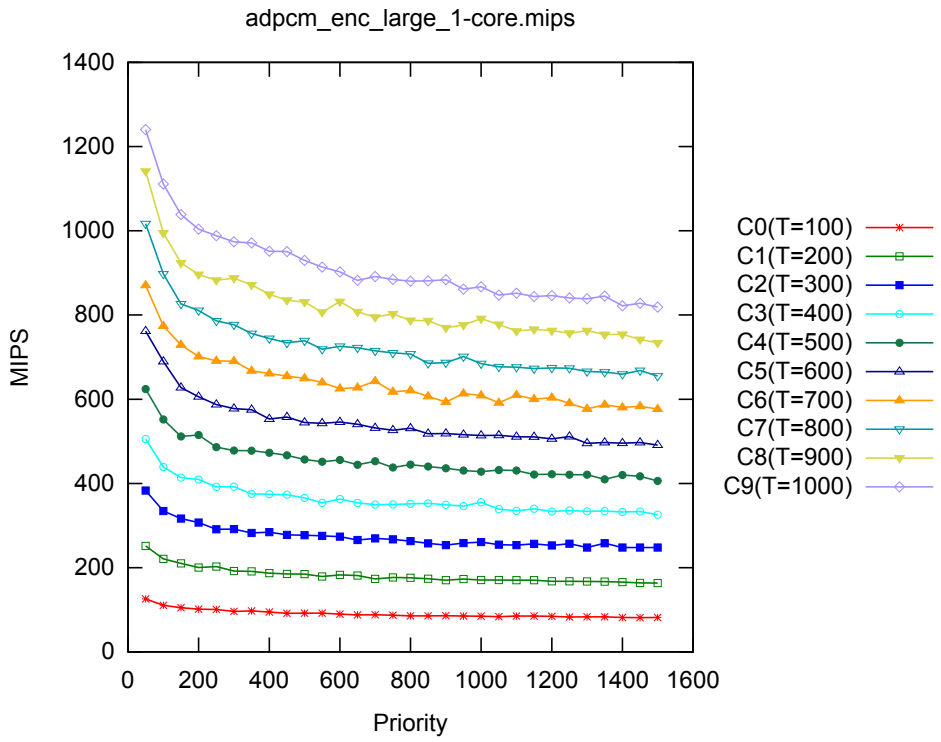


Figura 246: Desempenho em MIPS de Codificador ADPCM (Grande)

Na análise das estimativas de tempo simulado, considerando entrada de tamanho pequeno, foram geradas as curvas vistas na figura 247. O comportamento teórico previsto é observado, permitindo que um modelo de tempo

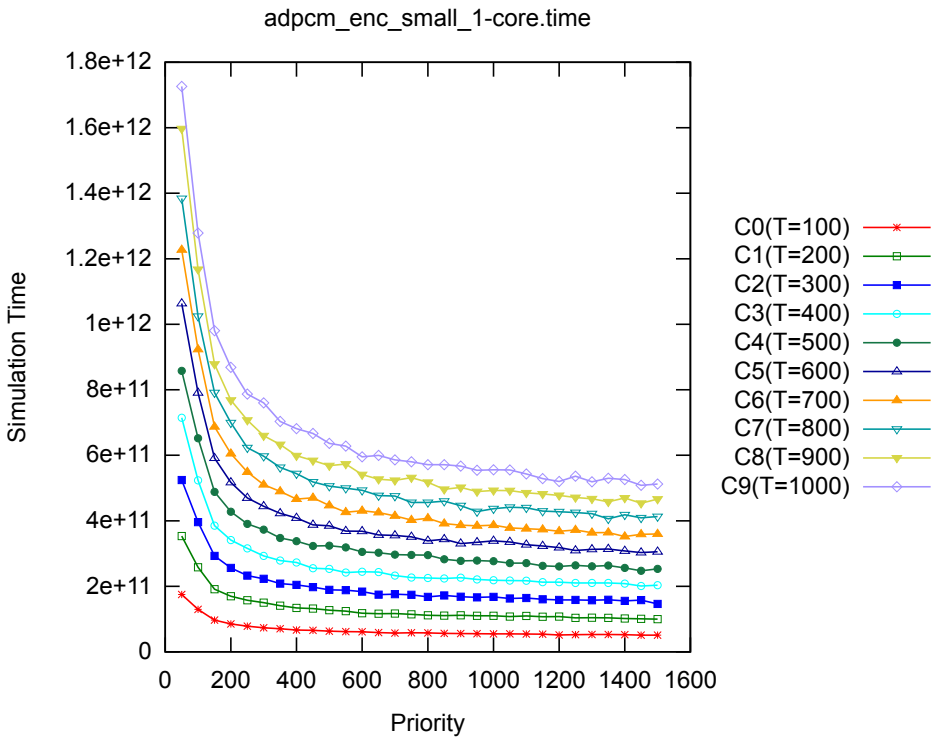


Figura 247: Tempo simulado de Codificador ADPCM (Pequeno)

preciso possa ser calculado e estimativas estáticas sobre o sistema possam ser realizadas. Na plataforma baseada em modelo ISS foi obtido um tempo simulado de 1,77698143e+11 picosegundos e desempenho de 1,27 MCPS e 0,64 MIPS. Para obter um comportamento análogo, foi calibrado no modelo proposto uma parametrização de ajuste e de prioridade com valores de 355 e 1.039, respectivamente, que geram uma estimativa de tempo simulado de 1,78902900796e+11 picosegundos em 6 simulações, com erro relativo ao ISS de 0,67% e desvio padrão de 1,882794368e+9 picosegundos (±1,05%). O desempenho obtido pelo modelo proposto foi de 140,77 MCPS e 281,53 MIPS que proporcionam um aumento na velocidade de simulação de cerca de 111 e 441 vezes, quando comparado ao desempenho obtido pelo ISS.

Quando a entrada de tamanho grande é utilizada na análise das estimativas de tempo geradas, um comportamento muito similar é observado no comportamento das curvas, com uma taxa de erro bem próxima da entrada pequena e manutenção das propriedades teóricas previstas no modelo. Na simulação utilizando um modelo ISS foi obtido um tempo simulado de 3,542018871e+12 picosegundos, com desempenho de 1,35 MCPS e 0,68 MIPS na execução da simulação. No modelo proposto foram calibrados parâmetros de ajuste e de priorização com valores 359 e 959, respectivamente,

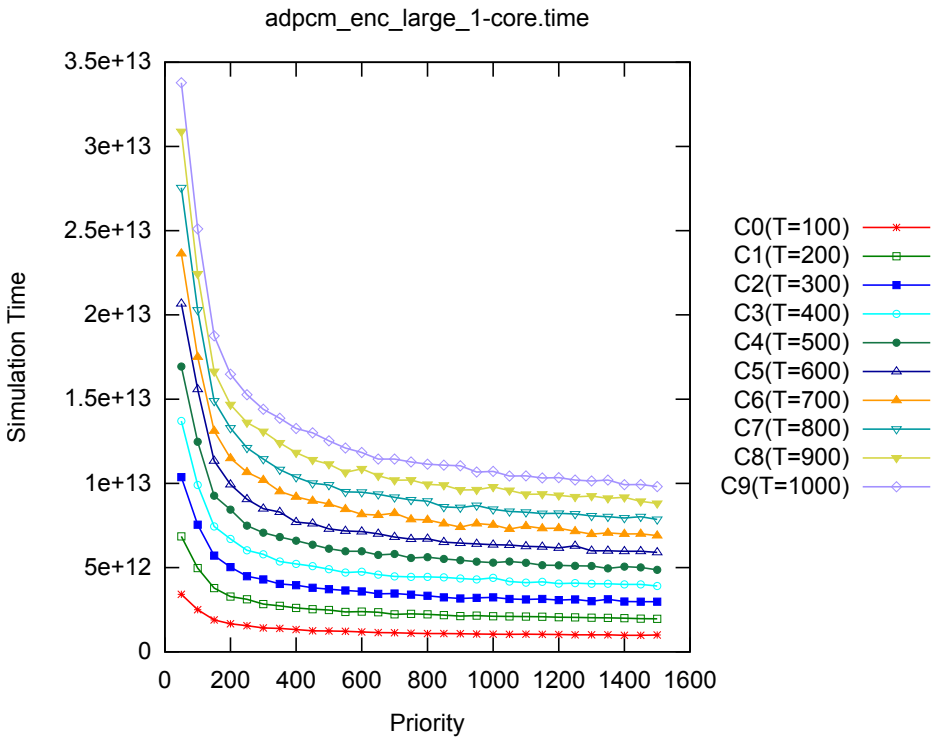


Figura 248: Tempo simulado de Codificador ADPCM (Grande)

para que um comportamento análogo seja observado na simulação. Foi obtida uma estimativa de tempo simulado de 3,513383306412e+12 picosegundos em 5 simulações, com desempenho de 143,48 MCPS e 286,95 MIPS. Estes resultados representam um aumento de desempenho de cerca de 106 e 425 vezes, respectivamente, com erro relativo ao ISS de 0,80% e desvio padrão de 3,98315826e+10 picosegundos (±1,13%).

### B.6.3 CRC32

A checagem de redundância cíclica ou Cyclic Redundancy Check (CRC) (90) é um código para detecção de erros muito utilizado em sistemas digitais de comunicação, permitindo que erros acidentais nos dados sejam detectados. Por sua grande importância e adoção em praticamente todos os sistemas de comunicação, uma implementação em software do CRC32 é fornecida no pacote de telecomunicações do MiBench, com duas entradas de tamanho pequeno e grande, fornecendo um conjunto de dados para que o valor do CRC seja calculado.

Definindo uma configuração de parâmetros de ajuste com 10 curvas (C0 até C9), com valores entre 100 e 1.000, com passos de tamanho 100, e de

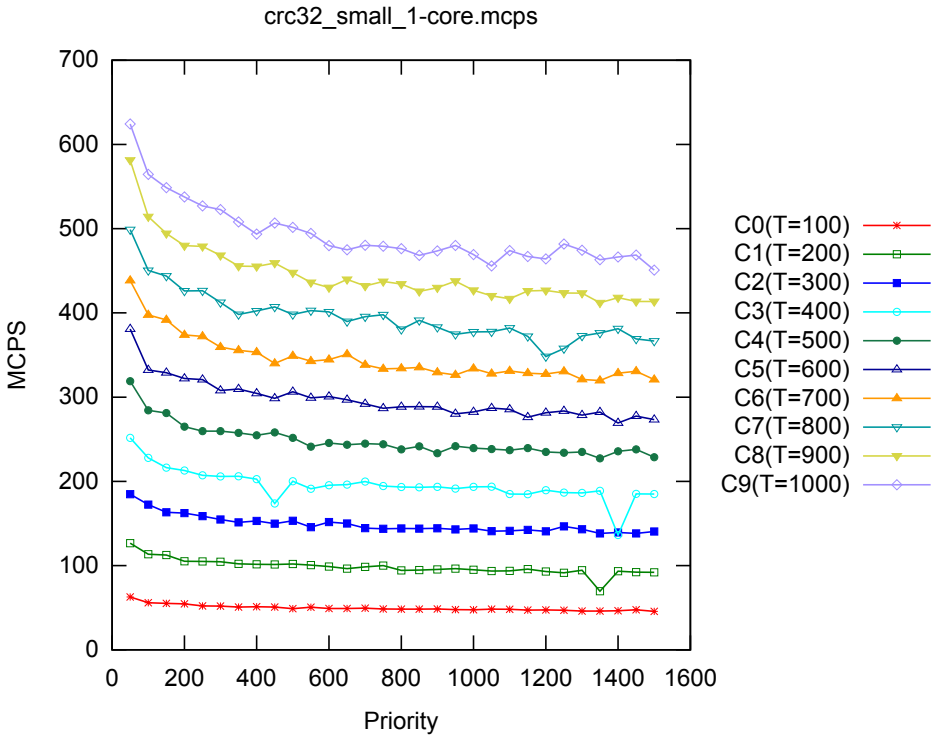


Figura 249: Desempenho em MCPS de CRC32 (Pequeno)

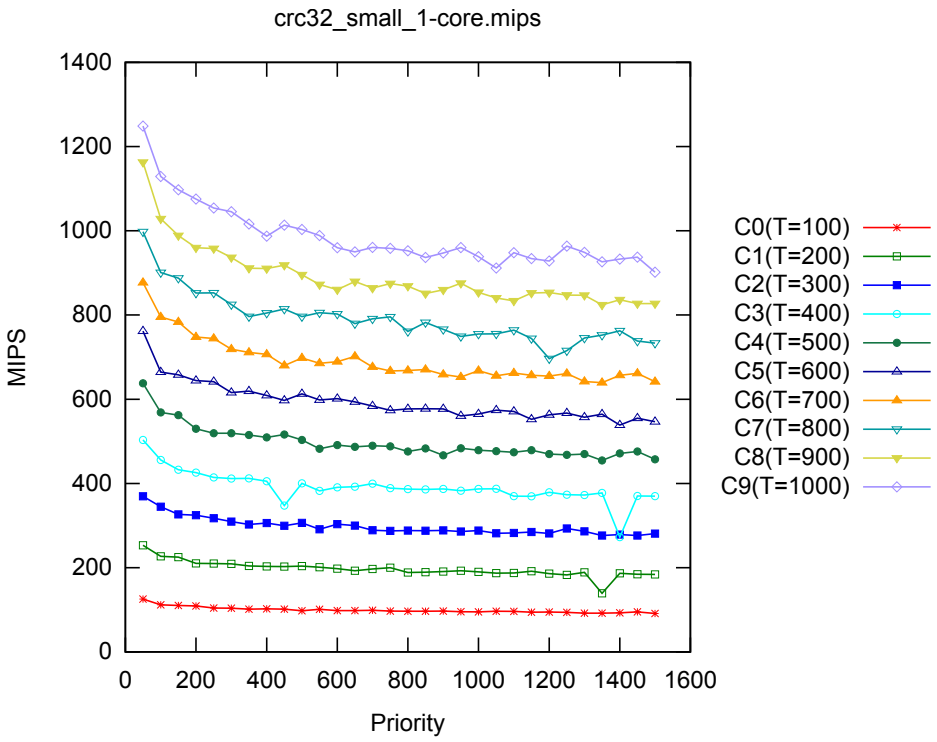


Figura 250: Desempenho em MIPS de CRC32 (Pequeno)

prioridade com valores entre 50 e 1.500, com intervalos de tamanho 50, são feitas análises de desempenho para entrada de tamanho pequeno em MCPS e MIPS, como pode ser visto nas figuras 249 e 250, respectivamente. No mo-

delo ISS foi obtido um desempenho de simulação de 1,35 MCPS e 0,65 MIPS, enquanto que no modelo proposto foram atingidos picos de desempenho de cerca de 600 MCPS e 1.200 MIPS que representam um aumento de velocidade de execução de aproximadamente 444 e 1.846 vezes, com relação ao ISS.

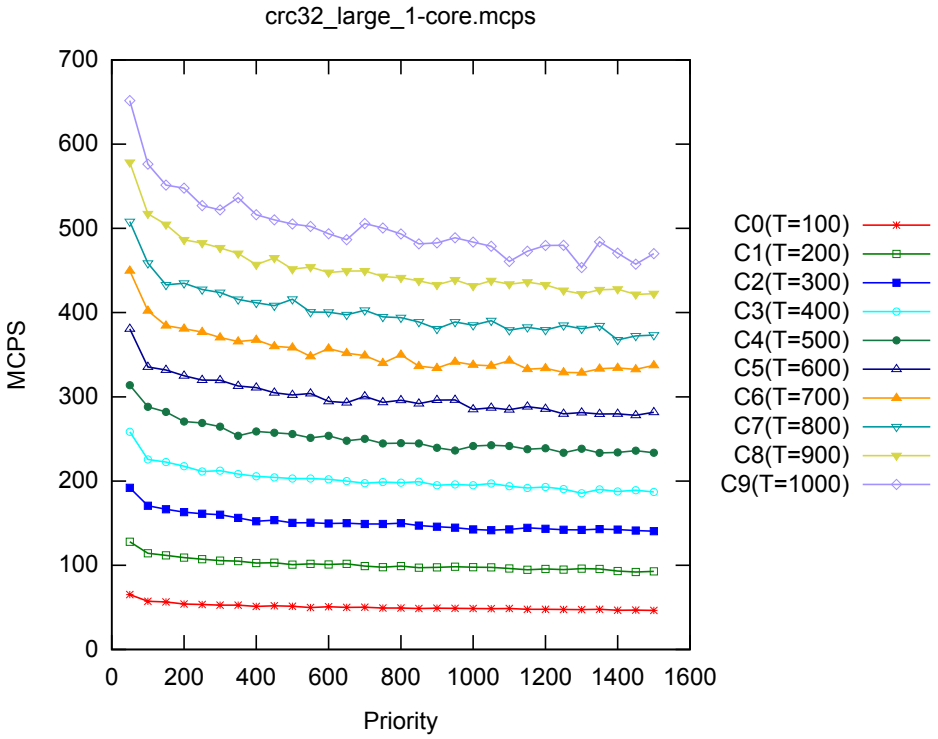


Figura 251: Desempenho em MCPS de CRC32 (Grande)

Utilizando a mesma parametrização descrita anteriormente, é feita a análise de desempenho com entrada de tamanho grande, sendo gerados os gráficos vistos nas figuras 251 e 252. Analisando os gráficos são vistos picos de desempenho de cerca de 650 MCPS e 1.300 MIPS no modelo proposto, representando uma melhoria de aproximadamente 492 e 2.031 vezes na velocidade de simulação, respectivamente, uma vez que no modelo ISS foi atingido um desempenho de 1,32 MCPS e 0,64 MIPS.

Na figura 253 podem ser vistas as estimativas de tempo simulado do CRC32 em sua execução com entrada de tamanho pequeno. É interessante notar que o comportamento teórico previsto é observado, apesar de ruídos nas curvas decorrentes do não determinismo da execução nativa da aplicação. Durante a execução do CRC32 com entrada pequena no ISS foi obtido um desempenho de 1,35 MCPS e 0,65 MIPS, com tempo simulado de 1,22509604e+11 picosegundos. Para equiparar este comportamento no modelo proposto são calibrados parâmetros de ajuste e de prioridade com valo-

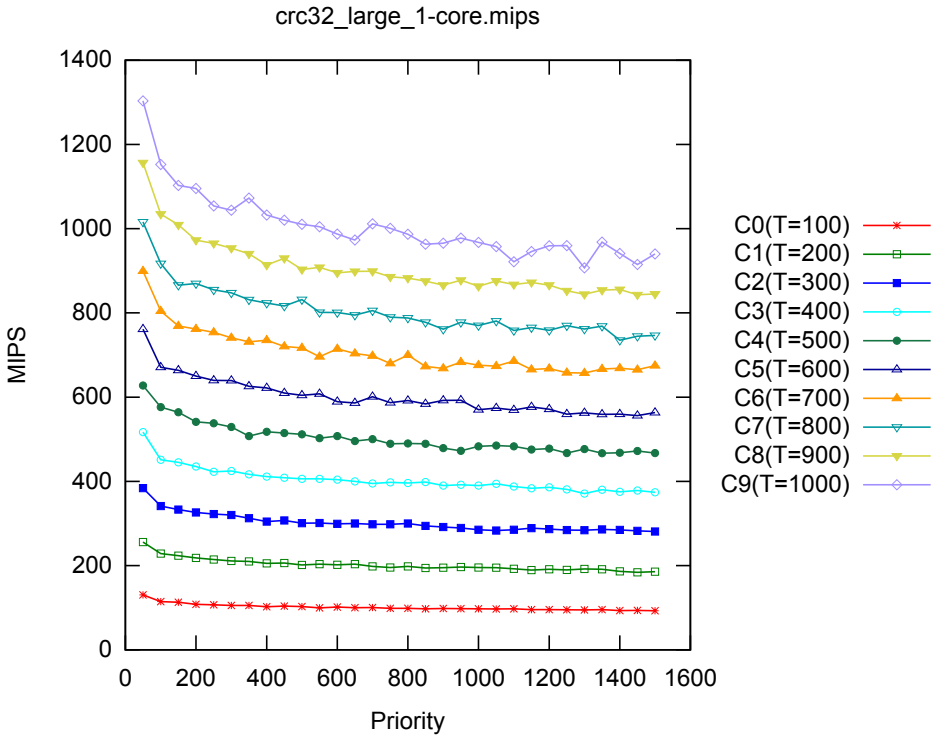


Figura 252: Desempenho em MIPS de CRC32 (Grande)

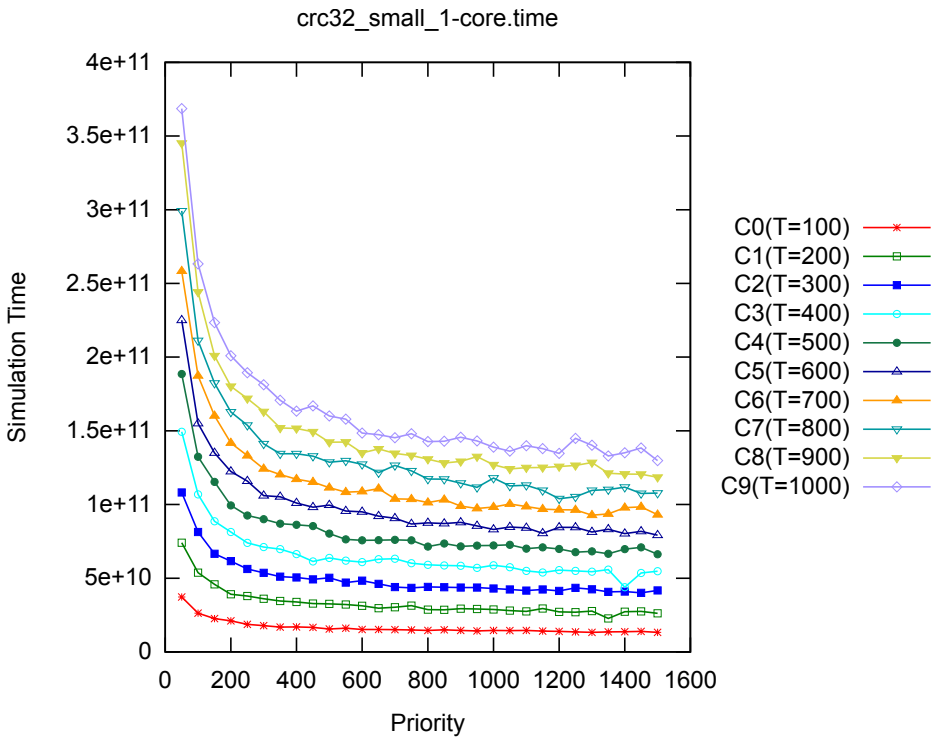


Figura 253: Tempo simulado de CRC32 (Pequeno)

res de 960 e 921, respectivamente, que geram uma estimativa de tempo simulado de 1,27885452828e+11 picosegundos em 29 simulações, com erro relativo ao ISS de 4,38% e desvio padrão de 3,075842273e+9 picosegundos (±2,40%).

Foi obtido um desempenho de 423,70 MCPS e 847,41 MIPS que aumentam a velocidade de simulação em cerca de 315 e 1.307 vezes, quando comparado ao desempenho do ISS.

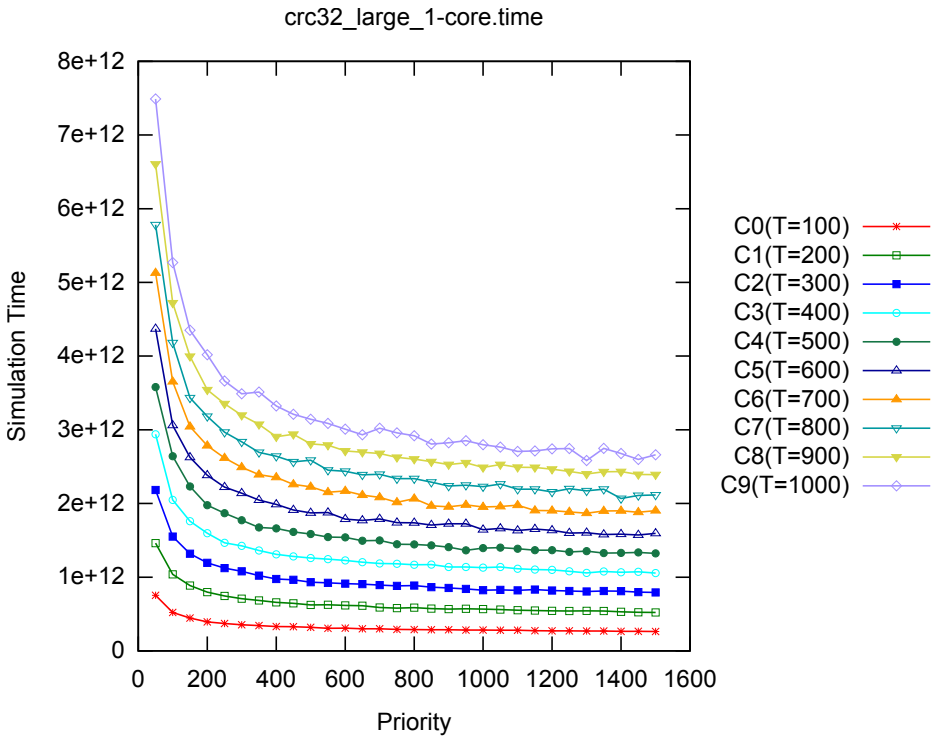


Figura 254: Tempo simulado de CRC32 (Grande)

Executando o CRC32 com entrada de tamanho grande, podem ser vistas na figura 254 as estimativas de tempo simulado geradas. Com o incremento do tamanho da entrada, mais iterações são realizadas pela aplicação e consequentemente mais amostras de tempo são capturadas pelo modelo proposto. Na simulação da plataforma com o CRC32 e utilizando um modelo ISS foi obtido um tempo simulado de 2,381367986e+12 picosegundos, com desempenho de 1,32 MCPS e 0,64 MIPS. Como é desejado obter um comportamento equivalente no modelo proposto, são calibrados parâmetros de ajuste e de priorização com valores de 932 e 974, respectivamente, gerando uma estimativa de tempo simulado de 2,40090104917e+12 picosegundos em 2 simulações, com erro relativo ao ISS de 0,82% e desvio padrão de 1,3981869607e+10 picosegundos (±0,58%). Com desempenho de 417,78 MCPS e 835,54 MIPS, o modelo proposto melhora a velocidade de simulação em cerca de 317 e 1.315 vezes, quando comparado ao ISS.

### B.6.4 FFT

A transformada rápida de Fourier ou FFT (91) é um algoritmo eficiente para calcular a transformada discreta de Fourier que é essencial em diversos sistemas de comunicação digital modernos. Por este motivo, o pacote de telecomunicações do MiBench possui uma implementação em software do algoritmo FFT, com dois modos de operação de tamanho pequeno e grande para demonstrar o funcionamento do algoritmo e realizar o cálculo dos coeficientes da transformada discreta de Fourier.

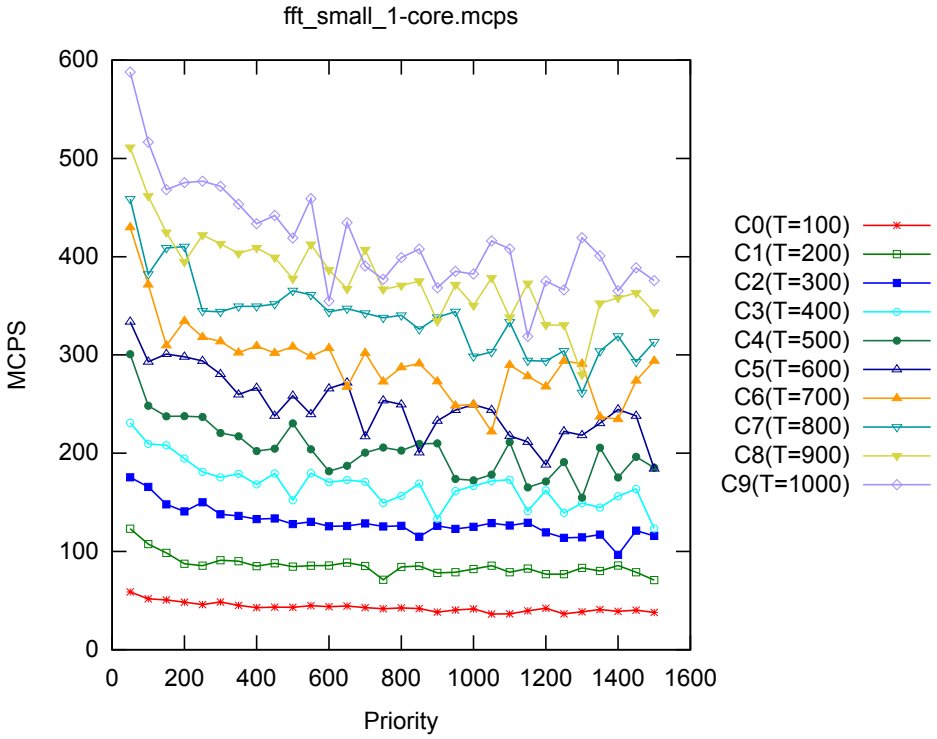


Figura 255: Desempenho em MCPS de FFT (Pequeno)

Definindo que os parâmetros de ajuste do modelo proposto formam 10 curvas (C0 até C9), com valores entre 100 e 1.000, com intervalos de tamanho 100, e prioridade entre 50 e 1.500, com passos de tamanho 50, são gerados os gráficos de desempenho vistos nas figuras 255 e 256, nas métricas MCPS e MIPS, respectivamente. Com pico de desempenho de cerca de 600 MCPS e 1.200 MIPS, o modelo proposto é aproximadamente 632 e 1.905 vezes mais rápido que o ISS que obteve 0,95 MCPS e 0,63 MIPS.

Com a mesma parametrização definida anteriormente, a execução de FFT com entrada de tamanho grande gera gráficos de desempenho que podem ser vistos nas figuras 257 e 258. Com desempenho um pouco superior, os resul-

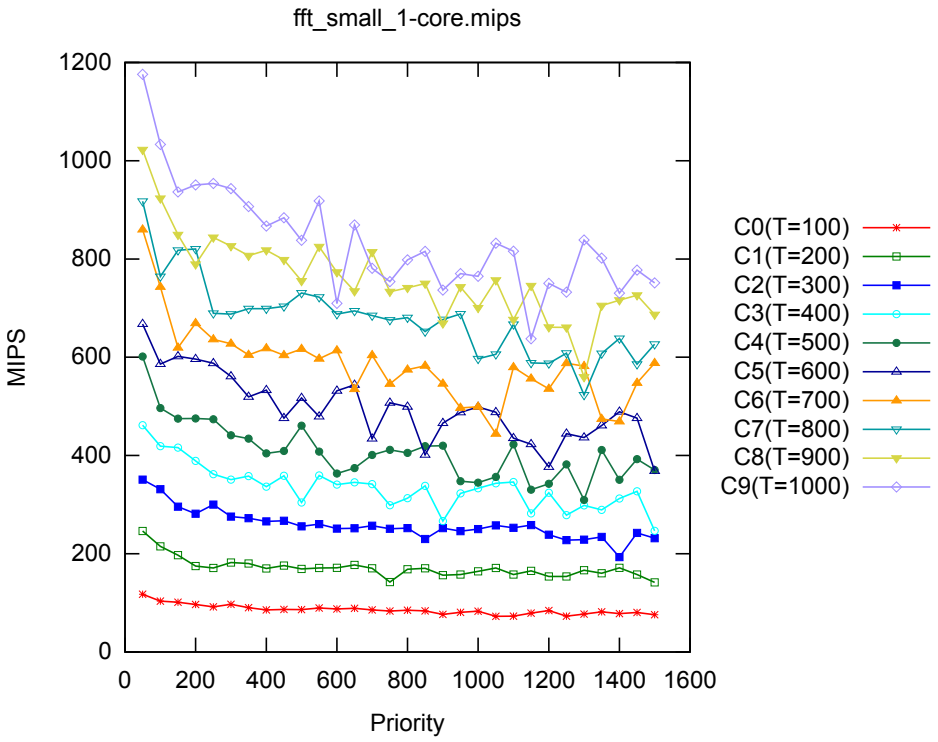


Figura 256: Desempenho em MIPS de FFT (Pequeno)

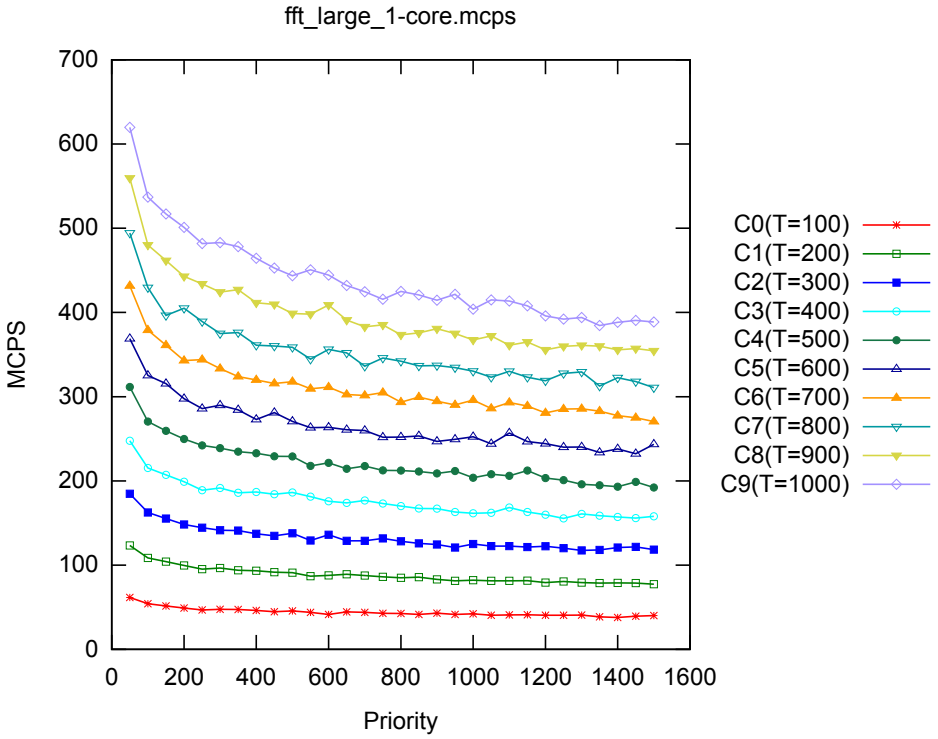


Figura 257: Desempenho em MCPS de FFT (Grande)

tados mostram picos de desempenho de aproximadamente 600 MCPS e 1.200 MIPS que tornam o modelo proposto cerca de 638 e 1.905 vezes mais rápido que o ISS que atingiu 0,94 MCPS e 0,63 MIPS.

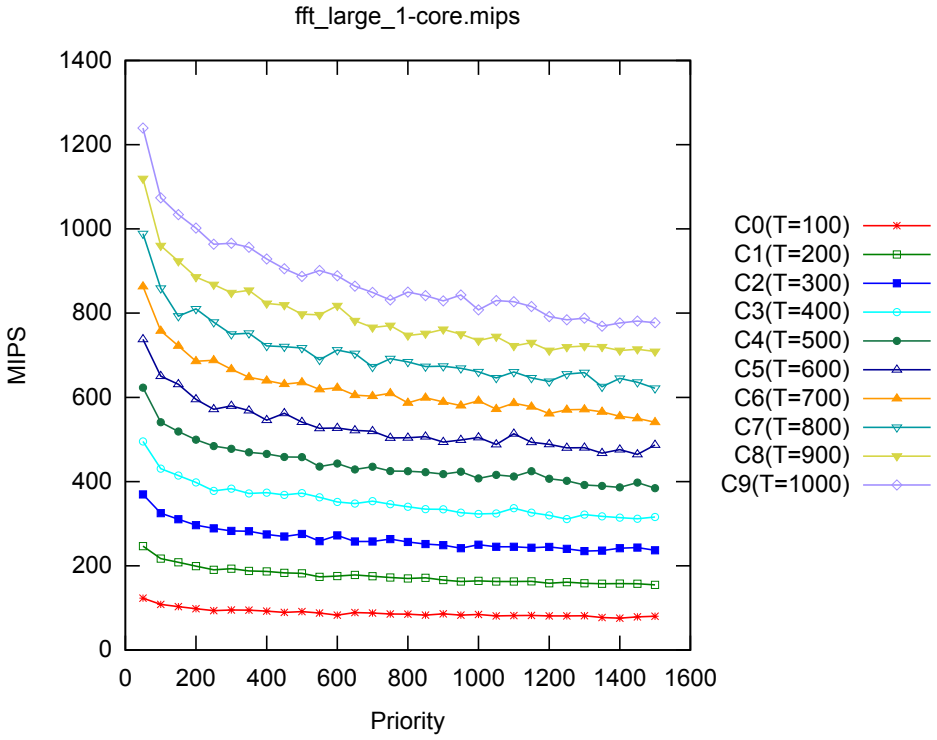


Figura 258: Desempenho em MIPS de FFT (Grande)

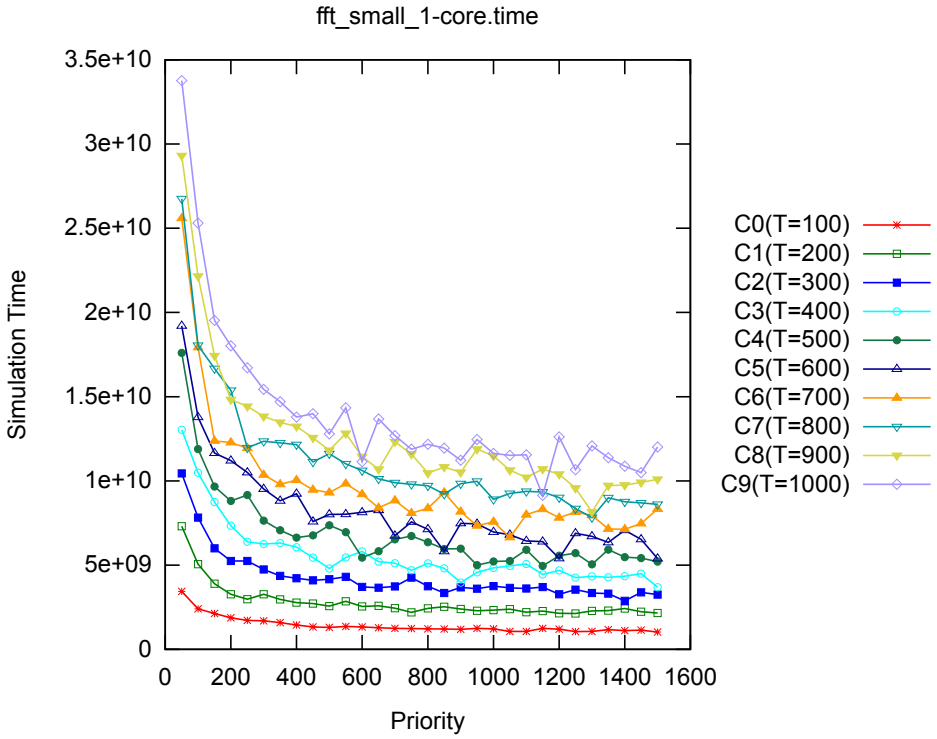


Figura 259: Tempo simulado de FFT (Pequeno)

Concentrando a atenção nas estimativas de tempo simuladas pelo modelo proposto, utilizando a entrada de tamanho pequeno, as curvas geradas podem ser visualizadas na figura 259. Neste gráfico é possível observar que

o comportamento teórico previsto é observado, mas que para valores maiores de ajuste o erro começa a crescer e reduzir a qualidade das previsões realizadas. Em uma plataforma baseada em ISS foi obtido um tempo simulado de 1,270241921e+12 picosegundos, com desempenho de execução de simulação de 0,95 MCPS e 0,63 MIPS. Para igualar o comportamento do ISS e medir o erro relativo da estimativa, é feita a calibração dos parâmetros de ajuste e de prioridade com valores de 135.465 e 938 que geram uma estimativa de tempo simulado de 1,377507483087e+12 picosegundos em 381 simulações, com erro relativo de 8,44% e desvio padrão de 1,16413946216e+11 picosegundos (±8,45%). O desempenho do modelo proposto foi de 42.520 MCPS e 85.040 MIPS que aumentam em cerca de 44.608 e 134.154 vezes a velocidade de simulação, quando comparado ao desempenho obtido com ISS.

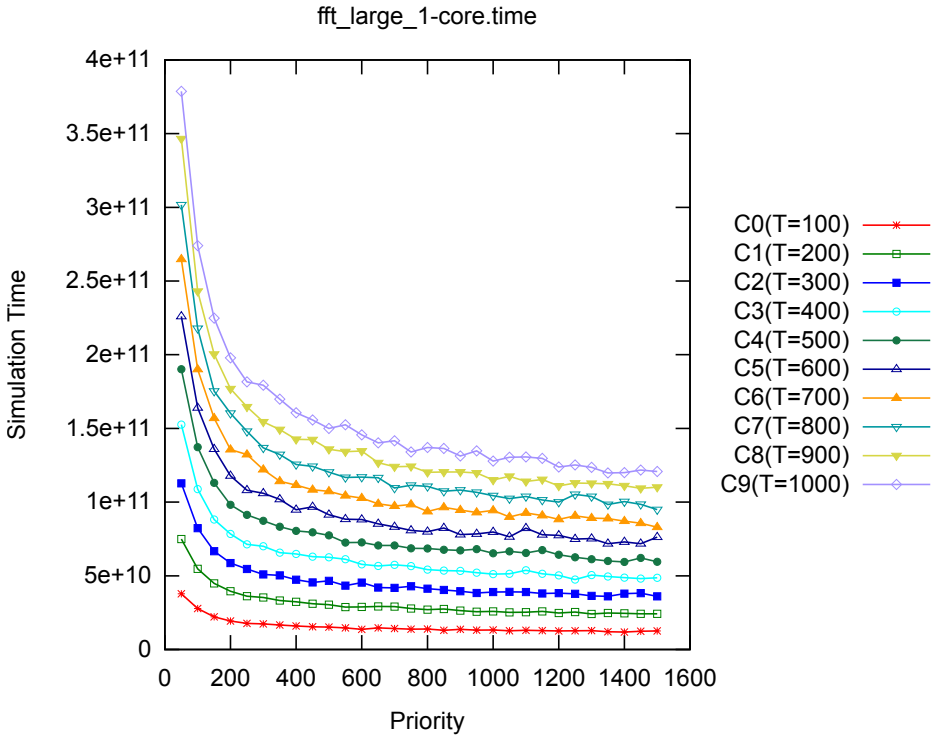


Figura 260: Tempo simulado de FFT (Grande)

Na figura 260 são exibidas as curvas de estimativas de tempo simulado da aplicação FFT com entrada de tamanho grande. Deve-se notar que o comportamento teórico foi aprimorado, pela redução dramática dos ruídos das estimativas e isto se deve ao fato da aplicação estar executando com mais iterações. O aumento do custo computacional tende sempre a melhorar a quantidade de amostragens de tempo realizadas e com isto reduzindo o erro associado. Na simulação com modelo baseado em ISS foi atingido um tempo

simulado de 2,5586422126e+13 picosegundos e o sistema executou com desempenho de 0,94 MCPS e 0,63 MIPS. Na configuração do modelo proposto foram utilizados parâmetros de ajuste e de prioridade com valores de 204.677 e 887, para obter um comportamento equivalente do sistema. Foi gerada uma estimativa de tempo simulado de 2,5836120662539e+13 picosegundos em 13 simulações, com desempenho de 80.119 MCPS e 160.239 MIPS. A estimativa gerada possui um erro relativo ao ISS de 0,97%, um desvio padrão de 4,88084310797e+11 picosegundos (±1,88%) e representa um aumento de desempenho de aproximadamente 84.854 e 253.342 vezes na execução do algoritmo FFT.

### B.6.5 IFFT

O algoritmo de transformada inversa rápida de Fourier ou IFFT realiza a transformação de coeficientes do domínio da frequência para o domínio do tempo. Assim como a FFT, o algoritmo de IFFT também está disponível no pacote de telecomunicações do MiBench, permitindo aos projetistas avaliar o desempenho de suas plataformas para executar operações básicas de processamento digital de sinais. São criados dois tipos de entrada de tamanho pequeno e grande para que seja feita a conversão dos coeficientes e os resultados calculados para cada entrada são exibidos no terminal de execução.

Para realizar os experimentos são definidos parâmetros de ajuste com 10 curvas (C0 até C9) com valores entre 100 e 1.000, com passos de tamanho 100, e prioridade com valores de 50 até 1.500, com intervalos de tamanho 50. Com esta parametrização são gerados os gráficos de desempenho vistos nas figuras 261 e 262, com métricas em MCPS e MIPS, respectivamente. Nestas simulações foi utilizada entrada de tamanho pequeno, com picos de desempenho de cerca de 600 MCPS e 1.200 MIPS que aumentam a velocidade de simulação em aproximadamente 645 e 1.936 vezes quando comparadas ao ISS que obteve 0,93 MCPS e 0,62 MIPS de desempenho.

Utilizando a mesma configuração de parâmetros definida anteriormente, são gerados os gráficos de desempenho para entrada de tamanho grande, como pode ser visto nas figuras 263 e 264. Os picos de desempenho observados chegam a aproximadamente 600 MCPS e 1.200 MIPS que aumentam a velocidade de simulação em cerca de 632 e 1.905 vezes em relação ao ISS

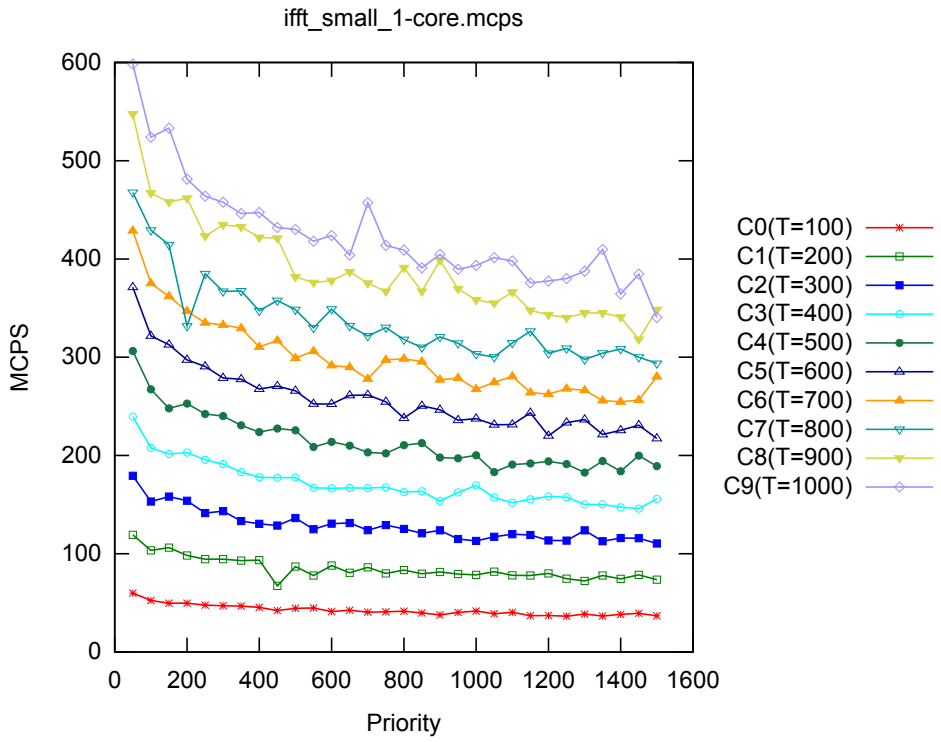


Figura 261: Desempenho em MCPS de IFFT (Pequeno)

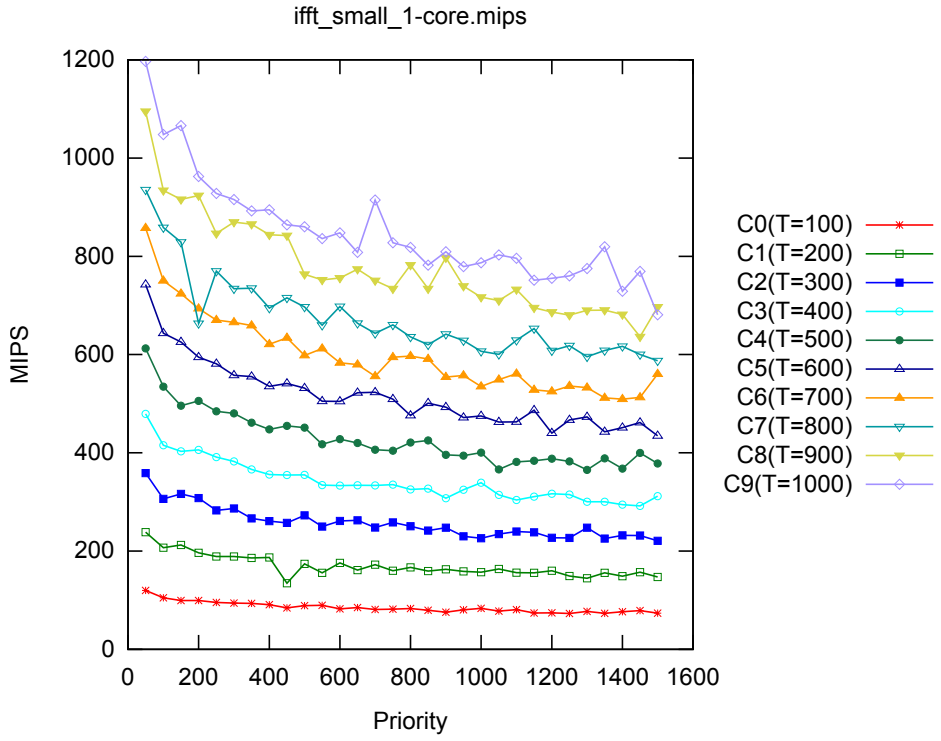


Figura 262: Desempenho em MIPS de IFFT (Pequeno)

que obteve 0,95 MCPS e 0,63 MIPS de desempenho.

Os resultados obtidos nas estimativas de tempo simulado com entrada pequena podem ser visualizados na figura 265. O comportamento teórico pre-

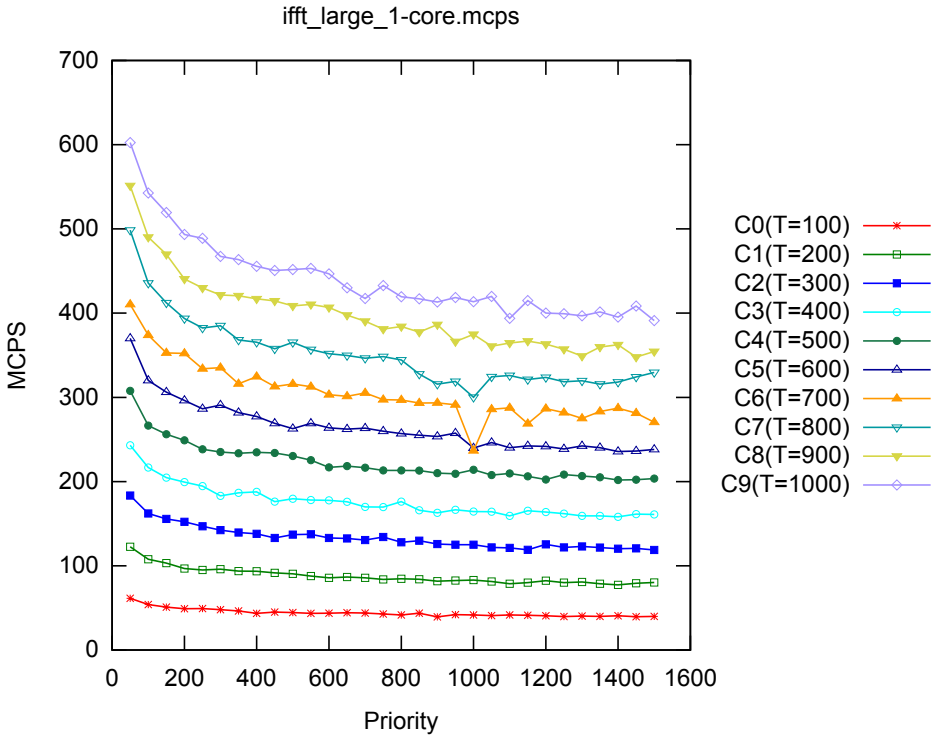


Figura 263: Desempenho em MCPS de IFFT (Grande)

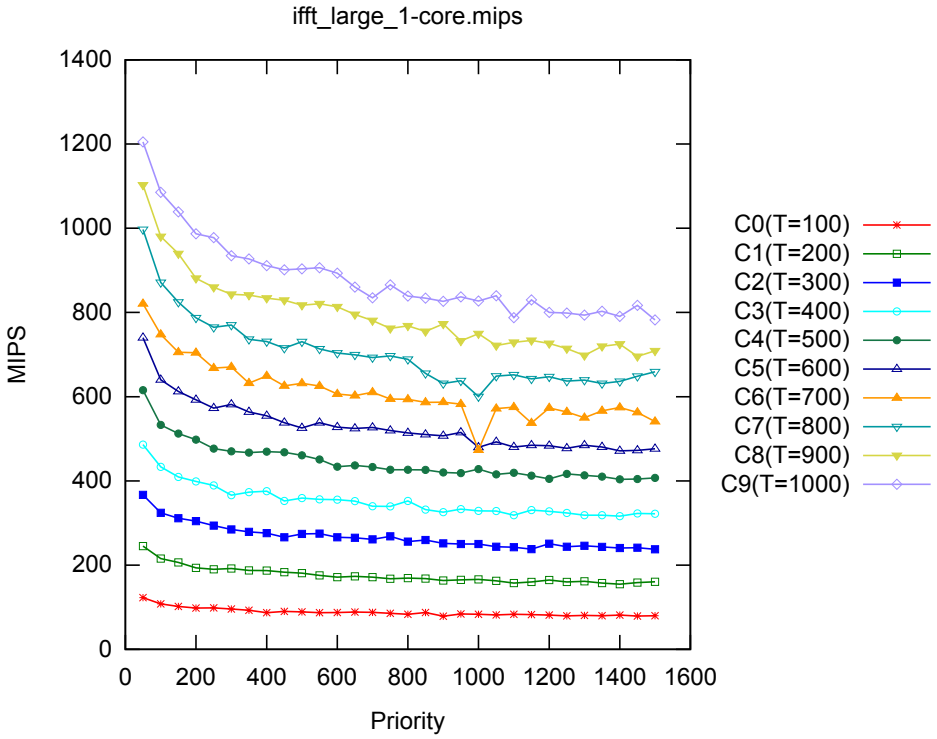


Figura 264: Desempenho em MIPS de IFFT (Grande)

visto é observado e alguns pontos fora da curva esperada são exibidos. Na realização da simulação do algoritmo IFFT na plataforma baseada em ISS foi obtido um tempo simulado de 3,077278146e+12 picosegundos e desempenho

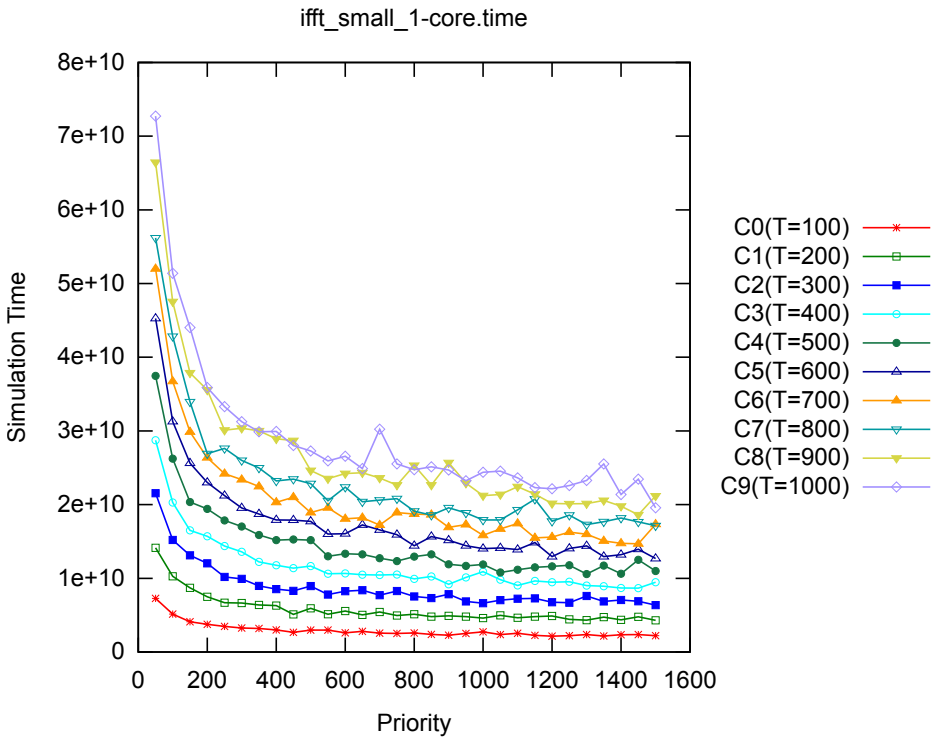


Figura 265: Tempo simulado de IFFT (Pequeno)

de simulação de 0,93 MCPS e 0,62 MIPS. Para obter um comportamento análogo no modelo proposto, são calibrados parâmetros de ajuste e de prioridade com valores de 130.138 e 862, respectivamente, que geram uma estimativa de tempo simulado de 3,146930591097e+12 picosegundos em 71 simulações, com erro relativo de 2,26% e desvio padrão de 1,3492358657e+11 picosegundos (±4,28%). Com desempenho de 50.375 MCPS e 100.749 MIPS houve um aumento de cerca de 54.120 e 162.263 vezes na velocidade de execução do algoritmo IFFT com relação a plataforma baseada em ISS.

As estimativas de tempo simulado para entrada de tamanho grande podem ser visualizadas na figura 266, utilizando a mesma configuração de parâmetros já descrita anteriormente. Como já foi dito, com o aumento do tamanho da entrada e consequentemente do número total de iterações da aplicação, a quantidade de amostras de tempo realizadas também é aumentada. Executando o algoritmo IFFT na plataforma baseada em ISS foi atingido um tempo simulado de 2,4944080539e+13 picosegundos e desempenho de execução da simulação de 0,95 MCPS e 0,63 MIPS. Na equiparação do comportamento no modelo proposto foram utilizados parâmetros de ajuste e de prioridade com valores 203.174 e 910, respectivamente, que geram uma estimativa de tempo simulado de 2,5072722572006e+13 picosegundos em 14

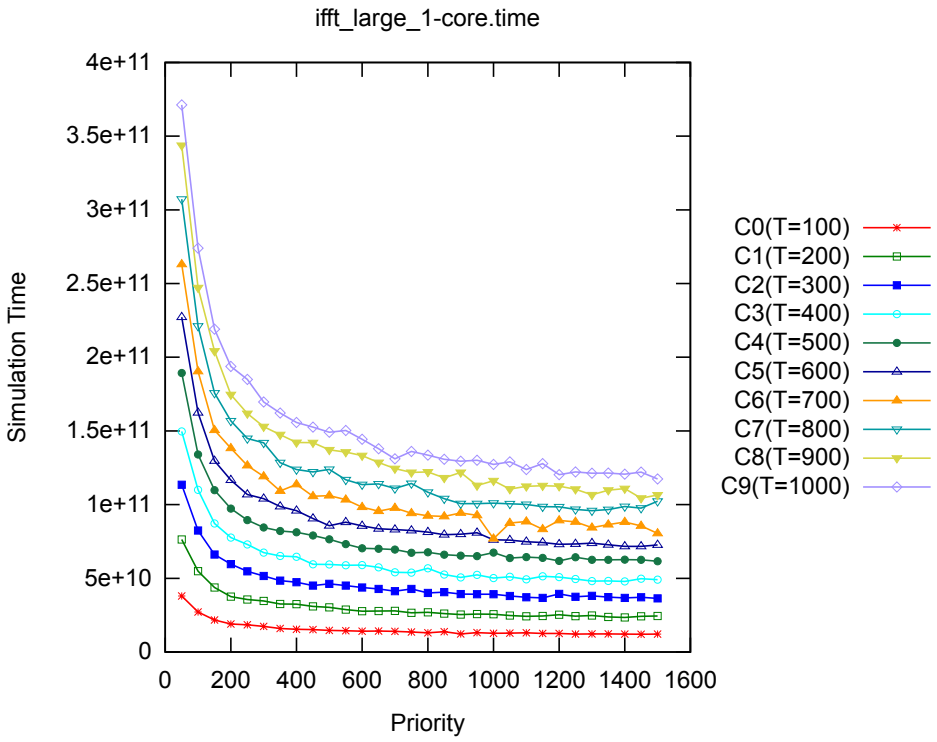


Figura 266: Tempo simulado de IFFT (Grande)

simulações. Com desempenho de 79.822 MCPS e 159.644 MIPS, houve um aumento de 84.486 e 252.322 vezes na velocidade de execução em relação ao ISS, com erro relativo de 0,51% e desvio padrão de 4,39237288842e+11 picosegundos (±1,85%).

### B.6.6 GSM Decoder

O Global System for Mobile Communications (GSM) (92) é um conjunto de padrões desenvolvidos pelo Instituto de Padrões em Telecomunicações Europeu (ETSI) e descreve a segunda geração (2G) de sistemas de telefonia móvel. O pacote de telecomunicações do MiBench possui uma implementação em software do decodificador GSM para que as funcionalidades necessárias sejam analisadas na plataforma de avaliação. Para o seu funcionamento são fornecidas entradas de tamanho pequeno e grande contendo um conjunto de dados codificados utilizando o padrão GSM para serem decodificados e restaurados em seu formato original.

Para realização dos experimentos serão feitas definições nos parâmetros de ajuste para gerar 10 curvas (C0 até C9), com valores entre 100 e 1.000, com intervalos de tamanho 100, e de prioridade com valores entre 50 e 1.500, com

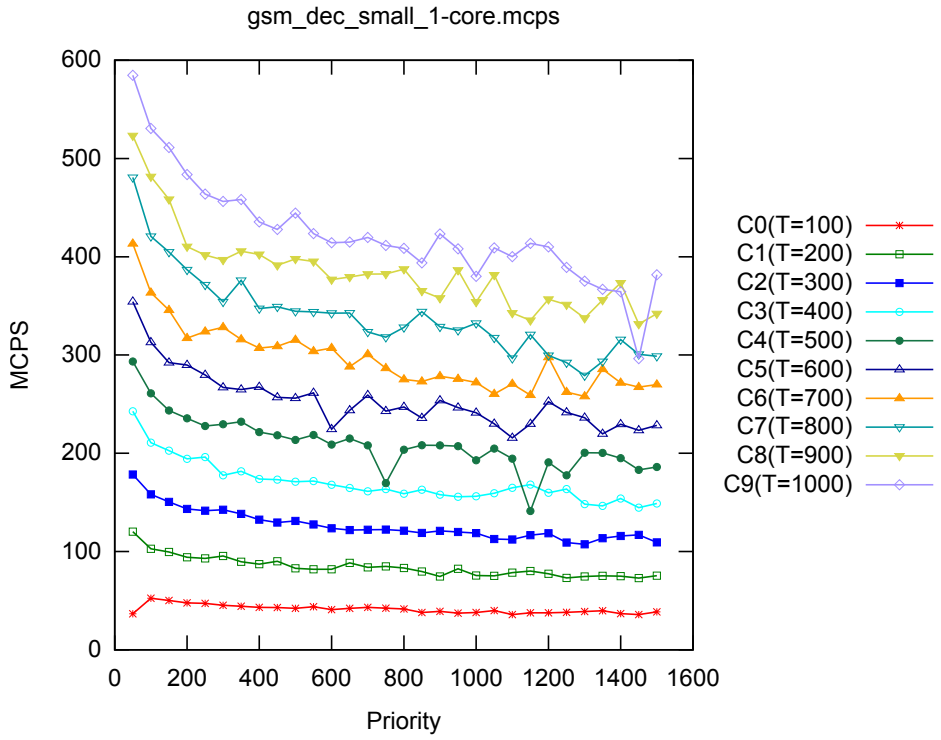


Figura 267: Desempenho em MCPS de Decodificador GSM (Pequeno)

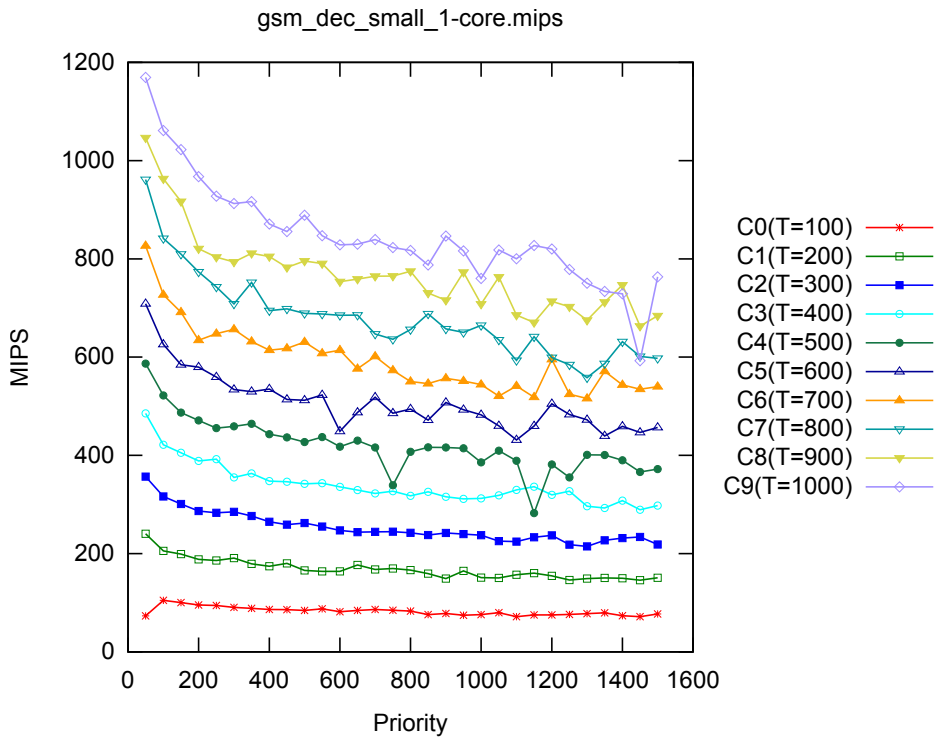


Figura 268: Desempenho em MIPS de Decodificador GSM (Pequeno)

passos de tamanho 50. Nos gráficos vistos nas figuras 267 e 268 são exibidas as curvas de desempenho em MCPS e MIPS, respectivamente, com execução do decodificador GSM com entrada de tamanho pequeno. Pode ser observado

um pico de desempenho de aproximadamente 600 MCPS e 1.200 MIPS que representam uma melhoria no desempenho de simulação de cerca de 561 e 1.875 vezes, tomando como referência a simulação com ISS que obteve 1,07 MCPS e 0,64 MIPS.

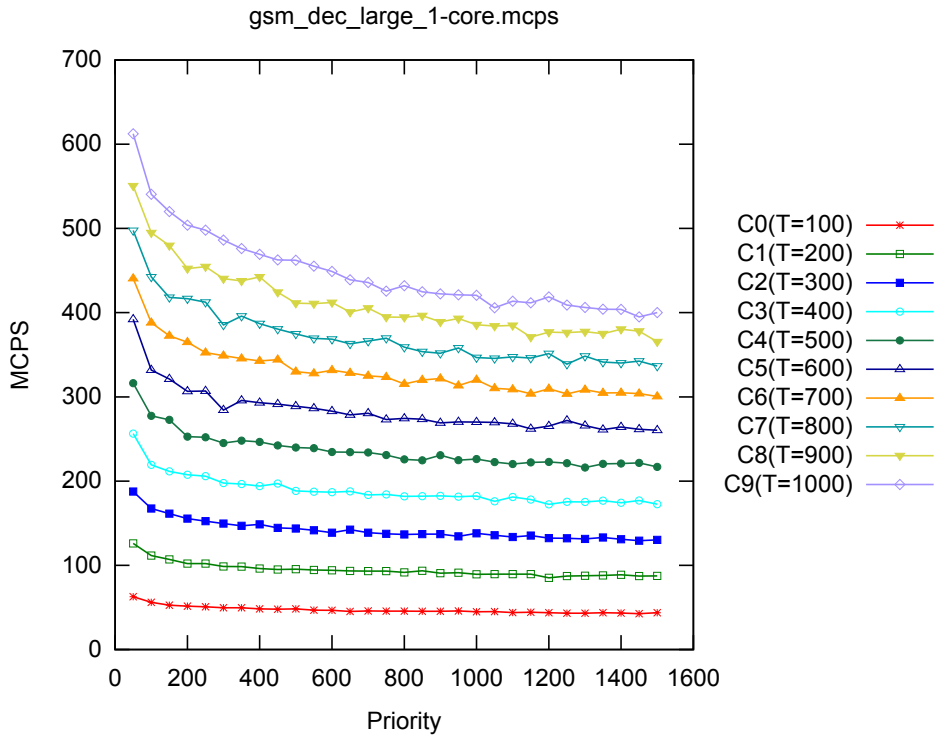


Figura 269: Desempenho em MCPS de Decodificador GSM (Grande)

Seguindo a configuração de parâmetros já definida, são gerados mais gráficos de desempenho para o decodificador GSM com entrada de tamanho grande, como pode ser visto nas figuras 269 e 270. Com pico de desempenho de cerca de 600 MCPS e 1.200 MIPS, o modelo proposto possui uma velocidade de simulação em torno de 541 e 1.818 vezes, respectivamente, mais rápida, quando em comparação com o modelo ISS que obteve 1,11 MCPS e 0,66 MIPS de desempenho.

Analisando as estimativas de tempo simulado com entrada pequena e mesma configuração de parâmetros são geradas as curvas ilustradas na figura 271. O comportamento observado reflete o que o modelo teórico proposto descreve e existem alguns ruídos associados ao não determinismo da execução nativa do software. Na simulação do decodificador GSM utilizando ISS foi obtido um tempo simulado de 7,6159752e+10 picosegundos, com um desempenho de 1,07 MCPS e 0,64 MIPS. Para que um comportamento equivalente seja obtido no modelo proposto, são calibrados parâmetros de ajuste

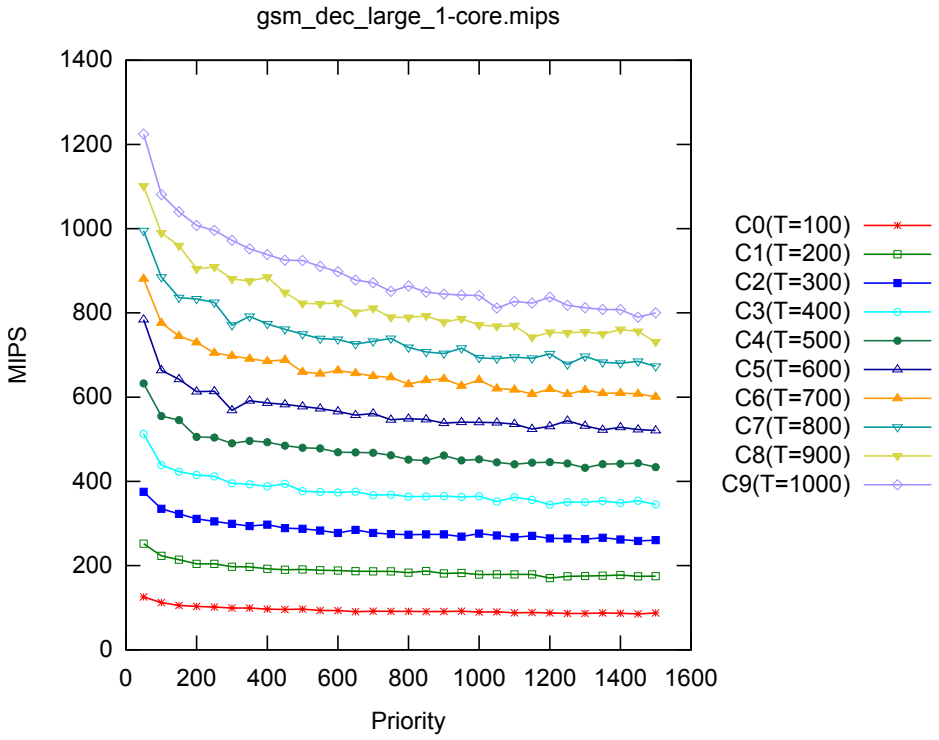


Figura 270: Desempenho em MIPS de Decodificador GSM (Grande)

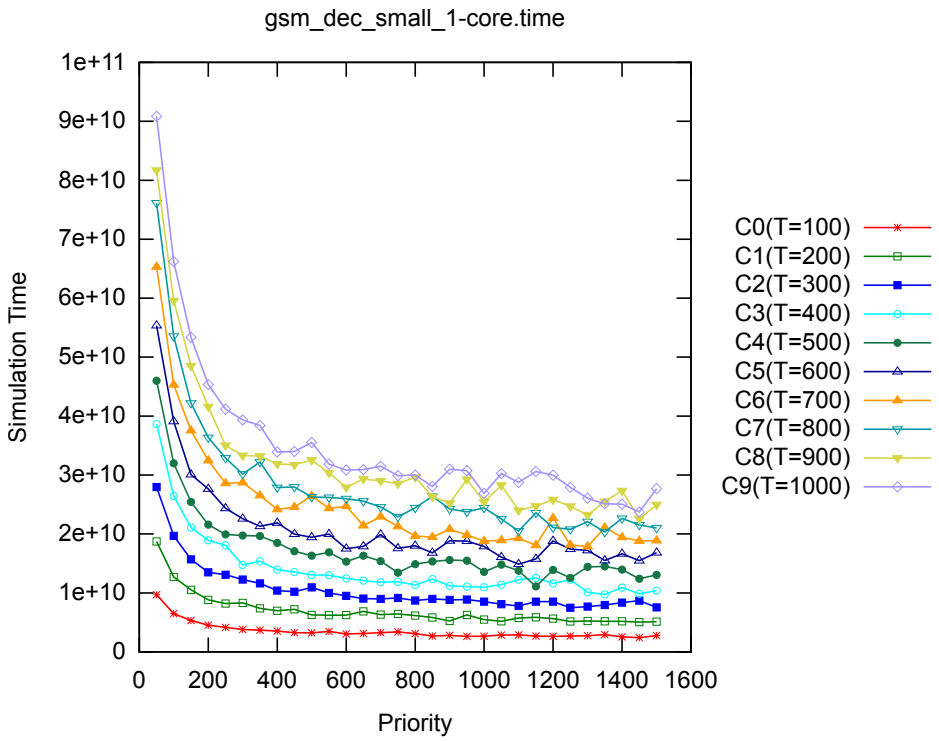


Figura 271: Tempo simulado de Decodificador GSM (Pequeno)

e de prioridade com valores de 2.835 e 1.053 que geram uma estimativa de tempo simulado de 7,9384022414e+10 picosegundos em 35 simulações, com erro relativo ao ISS de 4,23% e desvio padrão de 2,949824497e+9 picosegundos

(±3,71%). O modelo proposto apresentou desempenho de 1.129 MCPS e 2.258 MIPS que representam um aumento na velocidade de simulação de cerca de 1.058 e 3.528 vezes, respectivamente, em relação ao ISS.

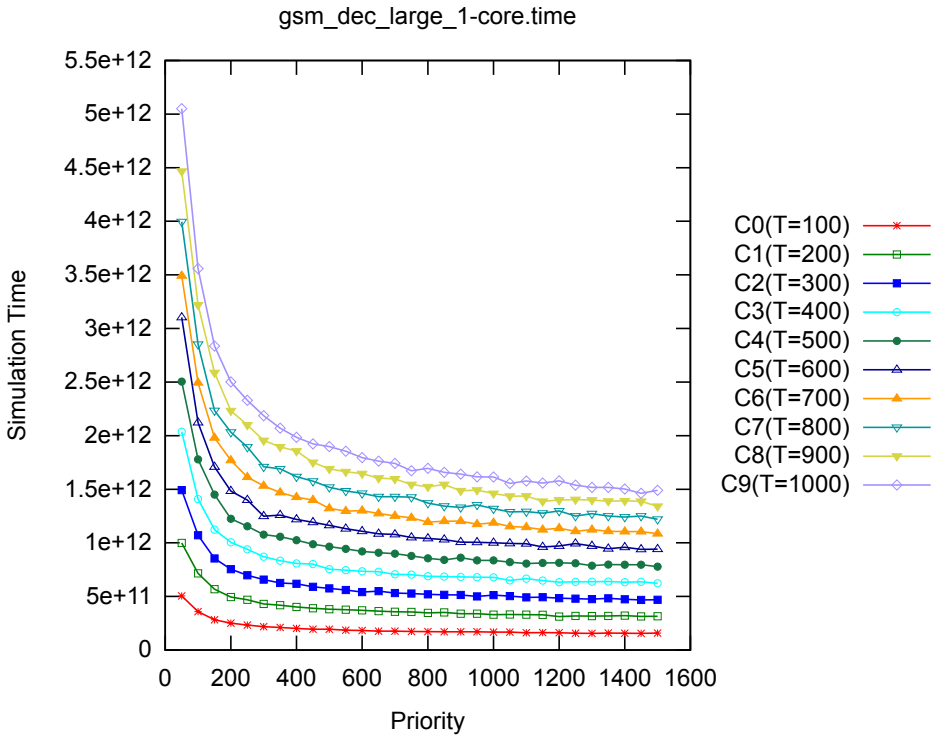


Figura 272: Tempo simulado de Decodificador GSM (Grande)

Para observar as estimativas de tempo simulado para entrada de tamanho grande são geradas as curvas ilustradas na figura 272. Conforme esperado, o aumento do tamanho da entrada torna as estimativas mais precisas e as curvas possuem um comportamento menos ruidoso. Na simulação do sistema com plataforma baseada em ISS foi obtido um tempo simulado de 4,117089637e+12 picosegundos, além de desempenho de 1,11 MCPS e 0,66 MIPS. Para equiparar este comportamento no modelo proposto são calibrados parâmetros de ajuste e de priorização com valores 2.641 e 938, respectivamente, que geram um tempo simulado de 4,125346106322e+12 picosegundos em 6 simulações, com erro relativo ao ISS de 0,20% e desvio padrão de 3,2297470856e+10 picosegundos (±0,78%). O desempenho atingido foi de 1.108 MCPS e 2.215 MIPS que representam uma melhoria de cerca de 1.003 e 3.348 vezes, respectivamente, do desempenho obtido com a simulação baseada em ISS.

### B.6.7 GSM Encoder

O pacote de telecomunicações do MiBench possui uma versão em software do codificador GSM para que sua implementação seja analisada em diferentes configurações de plataformas criadas, permitindo ao projetista avaliar o desempenho de sua solução para telecomunicação baseada em GSM. São fornecidas duas entradas de tamanho pequeno e grande para serem codificadas, sendo exibido na tela do terminal o resultado os dados codificados.

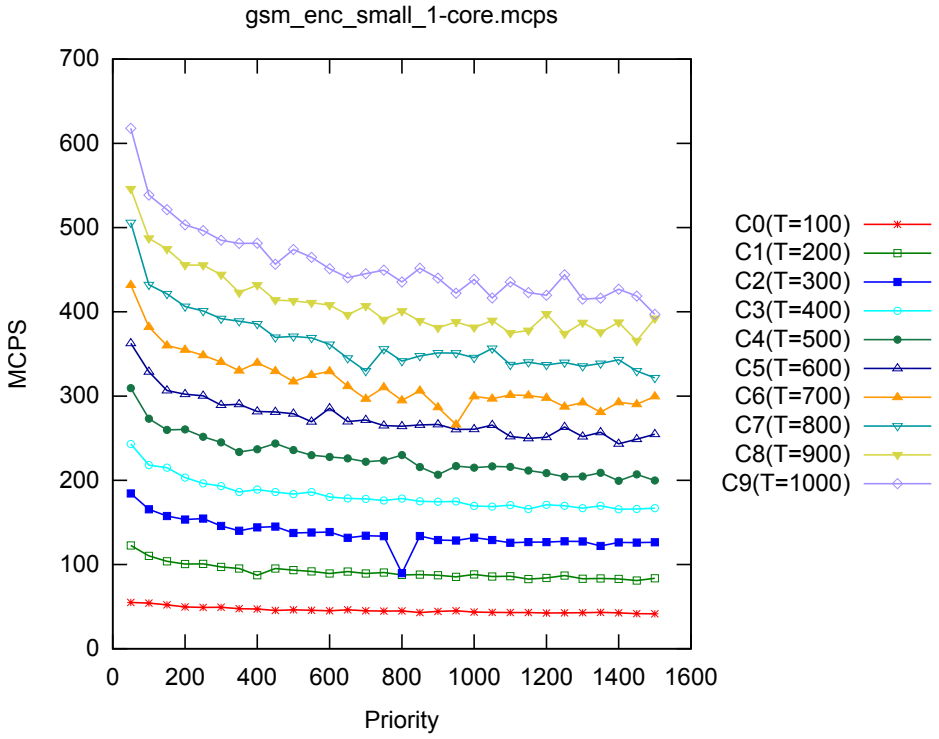


Figura 273: Desempenho em MCPS de Codificador GSM (Pequeno)

Para todos os experimentos realizados foi definida uma parametrização de ajuste com 10 curvas (C0 até C9), com valores entre 100 e 1.000, com passos de tamanho 100, e de prioridade com valores entre 50 e 1.500, com intervalos de tamanho 50. Nas figuras 273 e 274 são exibidas as curvas de desempenho em MCPS e MIPS, respectivamente, do codificador GSM com entrada de tamanho pequeno.

Pode ser visto um pico de desempenho com cerca de 600 MCPS e 1.200 MIPS que representa uma melhoria de aproximadamente 638 e 1.967 vezes, respectivamente, quando comparado ao desempenho do modelo ISS que foi de 0,94 MCPS e 0,61 MIPS.

Adotando a mesma configuração de parâmetros, são exibidas nas figuras

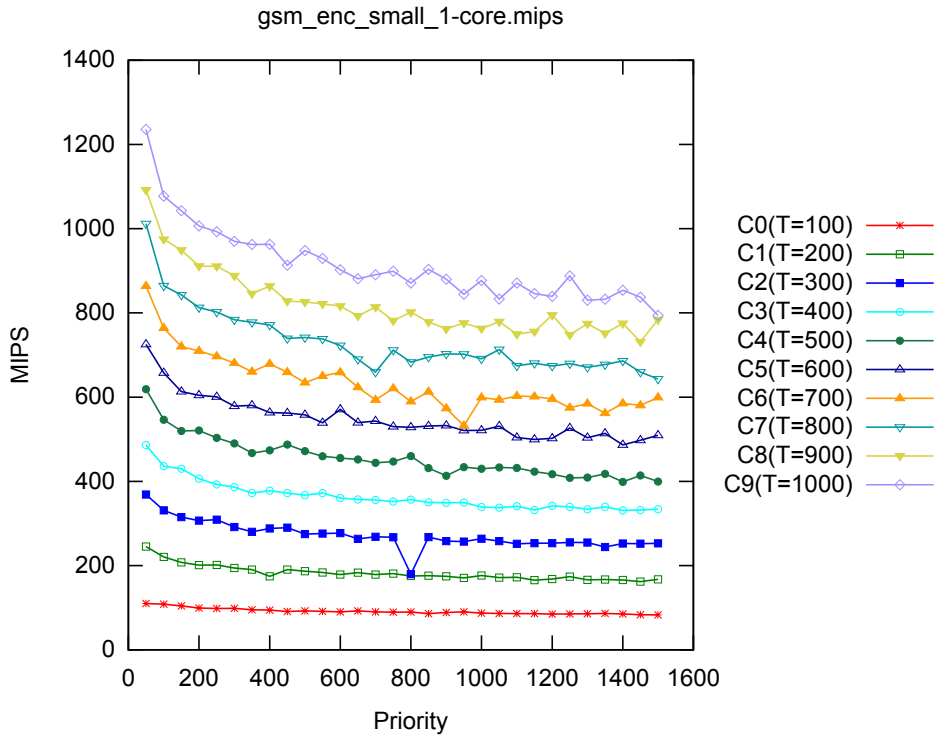


Figura 274: Desempenho em MIPS de Codificador GSM (Pequeno)

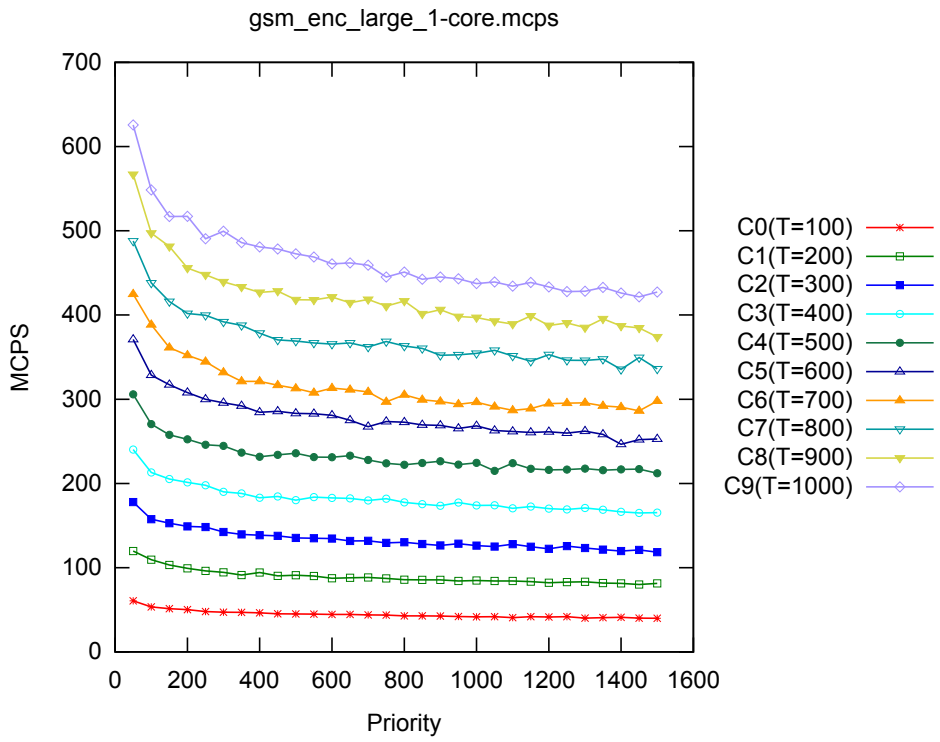


Figura 275: Desempenho em MCPS de Codificador GSM (Grande)

275 e 276 os desempenhos em MCPS e MIPS, respectivamente, para entrada de tamanho grande do codificador GSM. É possível perceber um pico de desempenho de aproximadamente 600 MCPS e 1.200 MIPS que aumentam em

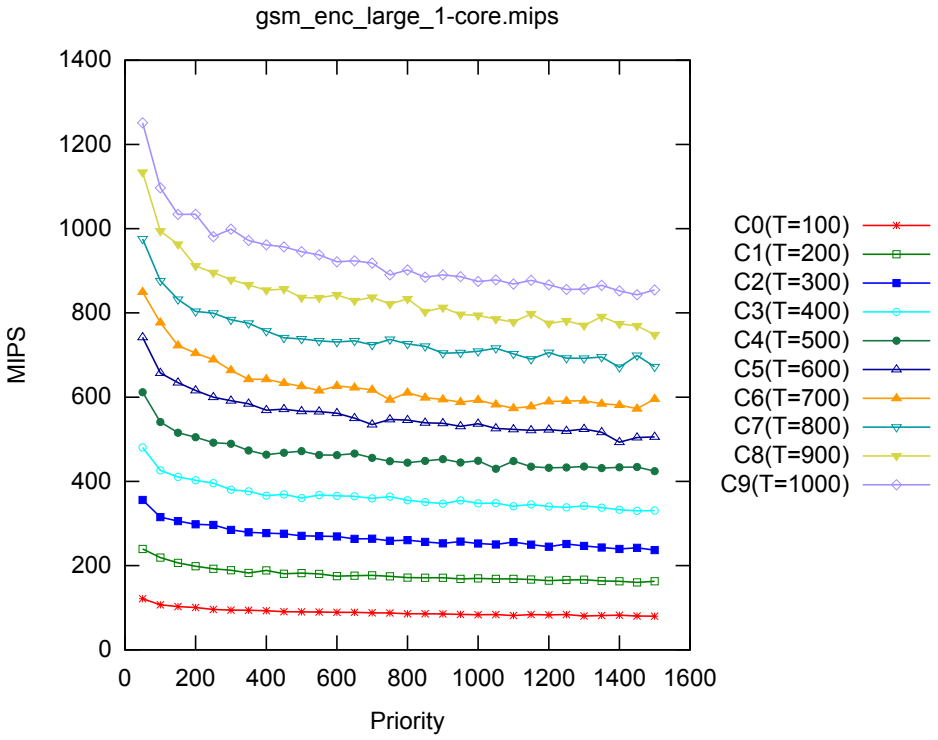


Figura 276: Desempenho em MIPS de Codificador GSM (Grande)

cerca de 619 e 1.905 vezes, respectivamente, a velocidade de simulação obtida pelo ISS de 0,97 MCPS e 0,63 MIPS.

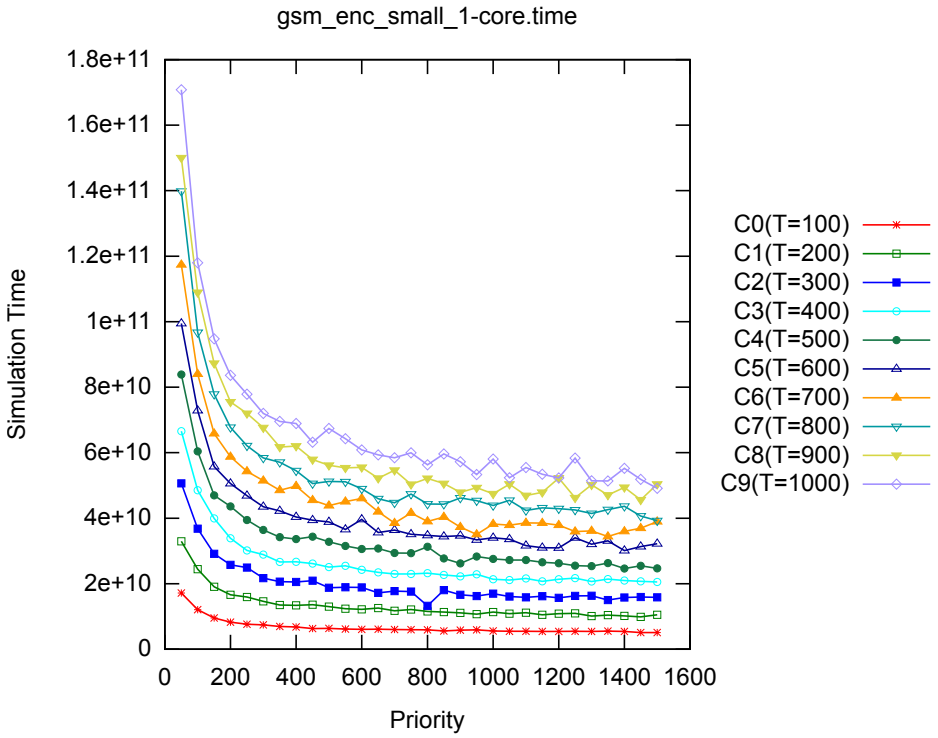


Figura 277: Tempo simulado de Codificador GSM (Pequeno)

Para realizar as análises das estimativas de tempo simulado com entrada

de tamanho pequeno, foi feita a geração das curvas vistas na figura 277. As estimativas obtidas confirmam o comportamento previsto no modelo teórico de tempo simulado e possuem um baixo erro associado, permitindo avaliação estática das configuração possíveis da plataforma. Utilizando um processador baseado em ISS foi obtido um tempo simulado de 2,44643291e+11 picosegundos, com desempenho de simulação de 0,94 MCPS e 0,61 MIPS. Para o modelo proposto obter um comportamento análogo ao ISS, foram configurados os parâmetros de ajuste e de prioridade com valores de 4.907 e 941 que geram um tempo simulado de 2,55846386913e+11 picosegundos em 31 simulações, com erro relativo ao ISS de 4,57% e desvio padrão de 7,907890789e+9 picosegundos (±3,09%). O desempenho do modelo proposto nesta configuração foi de 1.997 MCPS e 3.993 MIPS que o torna cerca de 2.126 e 6.534 vezes mais rápido que o modelo ISS.

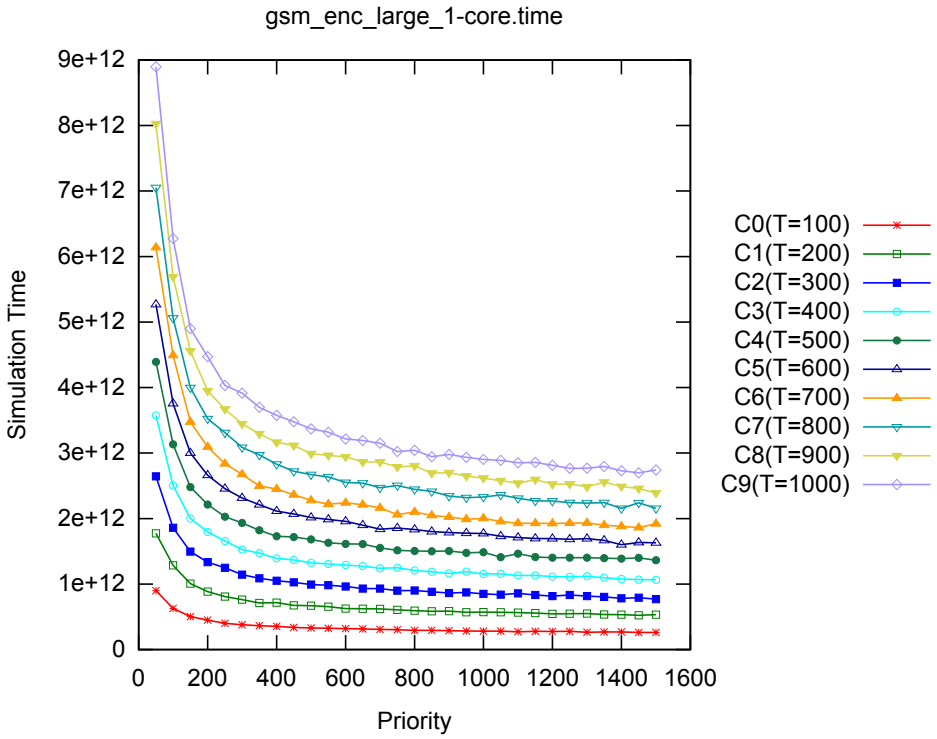


Figura 278: Tempo simulado de Codificador GSM (Grande)

Executando o codificador GSM com entrada de tamanho grande e gerando as curvas de estimativa de tempo simulado obtidas, é feita a geração do gráfico visto na figura 278. Com o aumento do tamanho da entrada, mais iterações são executadas pela aplicação e mais amostras de tempo de execução são coletadas e analisadas pelo núcleo de simulação. Em uma plataforma que utiliza um modelo de ISS para executar o software foi ob-

tido um tempo simulado de 1,3244937451e+13 picosegundos, com desempenho de simulação de 0,97 MCPS e 0,63 MIPS. O modelo proposto deve ser configurado com parâmetros de ajuste e de prioridade para ter um comportamento equivalente, sendo calibrados os valores 4.896 e 1.029, respectivamente. Com esta configuração foi obtida uma estimativa de tempo simulado de 1,3191822113643e+13 picosegundos em 4 simulações, com 2.035 MCPS e 4.070 MIPS e melhoria de cerca de 2.095 e 6.432 vezes no desempenho, com um erro relativo de aproximadamente 0,40% e desvio padrão de 9,2279470645e+10 picosegundos (±0,69%).